UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 3

Desarrollo de una herramienta que permita facilitar la toma de decisiones en el Proceso de Gestión de Cambios.

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Autores: Joel Jiménez Valido

Rayner Toledo Rios

Tutores: Ing. Dayanis Quintana Torres

Ing. Jorge Yuniel Jorrin Perdomo

Ciudad de la Habana, 2008-2009

Hay una fuerza motriz más poderosa que el vapor, la electricidad y la energía atómica: la voluntad.

Albert Einstein

Declaración de Auditoria

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los ____días del mes de _____ del año 2009.

_____

Joel Jimenez Valido

_____

Rayner Toledo Rios

_____

Ing. Dayanis Quintana Torres

_____

Ing. Jorge Yuniel Jorrin Perdomo

Datos del Contacto

Ing. Dayanis Quintana Torres

Ing. Jorge Yuniel Jorrin Perdomo

De Joel

A mis padres por tener fe en mí y su amor incondicional.

A mis amigos del alma Maikel, Kuki, Sergio, Tole (compañero de Tesis), Rega esta tesis es tuya también pa' que no haya drama.

A Malena por alimentarnos cocinera y amiga, Yaidel y Dany las veletas, Yanet, Elier, Javico, Rolo, Yurek el dominao.

Al Albe por ayudarme a entrar al proyecto y conseguir una maquina donde poder trabajar, a Ricardo corazon de leon por el salve con el netbeans, Al emo Chung Hibernate, Coriem por la Plantilla del ppt y todo los socios del proyecto borro el frio, pepone, eduardo, orno ,calvo y por ahí pa' allá.

A los otros compañeros de Tesis Dayanis y Jorrin.

Socios de la Facultad y todos los que han tenido que ver.

A todos mis educadores.

A la Revolución.

De Toledo

A mi madre, a mi padre por darme todo su apoyo, a mí querida mama por sus valiosos consejos, a mi esposa Jeanne por resistir tanto tiempo lejos de mí y a toda mi familia a mis abuelos, a mi tío, a mi tía y a todos mis primos porque de una forma todos contribuyeron a que hoy estuviera aquí

A mi amigo Jeison que es como un hermano para mí por estar en todo momento apoyándome.

A a todos mis amigos, a los buenos, a los de verdad…a Rega, Malena, Dany, Adrian, Roberto, Reinel, Maikel, Ququi, Yaidel, Yurek, PP, Javiko, Cori, Ornos, Yanet y todos los que no me acuerde ahora por compartir muchos momentos juntos a lo largo de estos 5 años que siempre recordare y por supuesto agradezco a mi compañero de tesis Nino, el mejor colega que he tenido que sin el no hubiera sido posible estar hoy aquí.

A mis tutores Dayanis y Jorrin por ayudarnos a poder realizar toda esta gran tarea.

A todos mis educadores.

Y a esta gran obra de la revolución que es la UCI.

De Joel

A mi mamá, papá, y sucia hermana, por siempre estar y hacerme lo que soy.

A mi mima y pipo los quiero mucho.

A abuela Zena.

A toda mi familia, primotes hermanos: Luiso, Maldi, Ale, Poli; Tias: Neli, Azule; y todos los demas porque mas cerca o lejos siempre estuvieron presente.

De Toledo

A mi mamá, mi papá, a yeyé, a mama, a mi hijo y a mi esposa y a toda mi familia y amigos.

Resumen

El objetivo de los proyectos productivos de la Facultad 3 es lograr una mayor eficiencia durante el proceso de producción de software y aunque las metodologías empleadas incluyen procesos de control, a lo largo del ciclo de desarrollo el sistema es víctima de innumerables cambios; en dependencia de la fase de desarrollo en que se encuentre el sistema, del tamaño y la complejidad del mismo será la dificultad con que el equipo de desarrollo podrá determinar el impacto que causará la realización de un cambio.

La presente investigación surgió por la necesidad de contar en los proyectos productivos de la Facultad 3 con una herramienta que permita al equipo de desarrollo tomar las decisiones necesarias para enfrentar un cambio.

El propósito de este trabajo es construir una herramienta sencilla, práctica y adaptable a cada equipo de trabajo y a cada desarrollador, pero que a su vez se rija por los estándares de Toma de Decisiones en la Gestión de Cambios, contribuyendo así con un ahorro de tiempo y recursos humanos para un mejor entendimiento entre los equipos de desarrollo. Esto ahorrará en gran medida el tiempo de construcción, maximizando la calidad en los productos de software.

Palabras Claves

Desarrollo de Software, Gestión de Configuración, Toma de Decisiones en la Gestión de Cambio

Contenido

Introducción

Antecedentes y problemática existente

El desarrollo alcanzado en la actualidad por las tecnologías de la Informática y las comunicaciones demanda cada día la construcción de software más potente que facilite el funcionamiento de las entidades.

Producto de la creciente complejidad de los sistemas que se construyen hoy en día los equipos de desarrollo se ven obligados a realizar actividades más complejas en función de garantizar la calidad del producto y cumplir sus compromisos con el cliente.

La Gestión de Configuración y, dentro de esta la Gestión de Cambios es una de las actividades que realizándose correctamente contribuye a incrementar la calidad del producto deseado ya que esta proporciona un mecanismo de control y seguimiento de los cambios que sean necesarios realizar al proyecto sin importar en qué momento de su ciclo de vida se encuentre.

La Facultad 3 de la Universidad de las Ciencias Informáticas (UCI), cuenta con un grupo de proyectos productivos que aportan al país una considerable suma de dinero desde hace poco más de 3 años con la exportación de software principalmente hacia la República Bolivariana de Venezuela, en la facultad se ha ido ganando en prestigio paulatinamente con los productos que en ella se desarrollan, pero no queda exenta del problema que constituye la gestión de cambios, ya que la misma cuenta en sus proyectos productivos con un procedimiento definido para realizar los cambios pero en la mayoría de los casos la puesta en práctica de dicho procedimiento se torna difícil y en algunos casos casi imposible ya que realizar cada una de las actividades que este posee de forma manual resulta muy complejo.

En entrevistas realizadas en los proyectos productivos de la Facultas 3 han podido constatarse las deficiencias que existen al enfrentar la Gestión de Cambios (Ver Anexo 1):

- El 40% de los proyectos productivos de la Facultad 3 no realiza la Gestión de Cambio.
- De los proyectos que realizan la Gestión de Cambio el 90% cuenta con un procedimiento definido para realizar esta actividad.
- De los proyectos que cuentan con un procedimiento definido para Gestionar los Cambios el 75% realiza la actividad de Análisis del Impacto de forma empírica y el 25% utiliza alguna técnica seleccionada por el equipo de desarrollo.

- Los proyectos productivos de la Facultad 3 carecen de herramientas para La Toma de Decisiones en la Gestión de Cambios ya que los que existen no definen la manera en que debe analizarse el impacto de los cambios.

Por tanto la investigación se orienta para resolver el siguiente **Problema Científico Investigativo:** El procedimiento definido para realizar la toma de decisiones en la gestión de cambios en los proyectos de la Facultad 3 aumenta excesivamente la carga de trabajo al equipo de Gestión de Configuración producto de que las actividades que él propone son altamente complejas.

Para ello se plantea como **Objetivo General:** Desarrollar una herramienta Informática que permita automatizar el procedimiento para la Toma Decisiones durante el proceso de Gestión de Cambios.

Objeto de estudio

El proceso de Gestión de Configuración.

Campo de acción

Proceso de Gestión de Cambios en los proyectos productivos de la Facultad 3.

Hipótesis

Si dotamos a los equipos de desarrollo de una herramienta informática que permita automatizar la Toma de Decisiones durante la Gestión de Cambios aseguraremos que dicha actividad se realice con la calidad requerida en cada uno de los proyectos productivos de la Facultad 3.

Objetivos Específicos

1. Realizar el estudio del estado del arte de los procedimientos para llevar a cabo la Toma de Decisiones durante el proceso de Gestión de Cambios.
2. Realizar el estudio del estado del arte de herramientas para llevar a cabo la toma de decisiones en la Gestión de Cambios.
3. Realizar el estudio del estado del arte de las metodologías, lenguajes y herramientas a utilizar para el desarrollo del sistema.
4. Realizar la Planificación y el Diseño del Sistema.
5. Realizar la Implementación y Pruebas del Sistema.
6. Realizar la validación del sistema.

Métodos científicos utilizados

Para la realización del trabajo se tienen en cuenta algunos métodos tradicionales investigativos. A continuación se mencionarán cada uno de ellos y de qué forma se ponen de manifiesto en la investigación.

Los métodos teóricos aplicados en la investigación son los métodos histórico-lógico puesto que se realizó un estudio de las tecnologías que existen actualmente para poder realizar la selección de las que se van a utilizar de acuerdo a las características propias del sistema a desarrollar y el método de análisis y síntesis, pues se realiza un análisis de la bibliografía que se utilizó para el estudio del tema.

Los métodos empíricos son los que describen y explican las características fenomenológicas del objeto, representan un nivel de la investigación cuyo contenido procede de la experiencia y es sometido a cierta elaboración racional.

El método empírico que se utiliza es el método de la observación, con la aplicación del mismo se puede conocer la realidad mediante la percepción directa de los objetos y fenómenos; a través de este método se pudo conocer la esencia de la problemática definida, lo que ayudó al planteamiento del problema científico, además de que permite conocer el proceso definido como objeto de estudio, lo cual influye a la hora de tener un conocimiento más detallado de lo que se quiere, lo que hace falta hacer y cómo hay que hacerlo.

Estructuración de la tesis

Capitulo 1: Fundamentación Teórica.

Se introducen los principales conceptos y términos empleados en la Gestión de Configuración de Software, la Gestión de Cambios y el procedimiento utilizado por la facultad 3, también se describen las posibles tecnologías y metodologías que podrán ser utilizadas para la solución del problema planteado.

Capitulo 2: Gestión y Diseño del Sistema.

Se describen brevemente las funciones principales y el flujo actual de los procesos involucrados en el problema en cuestión, haciéndose un análisis crítico de cómo se ejecutan actualmente estos procesos.

Además en el desarrollo de la aplicación se detallarán los pasos de la metodología que se propone en el capítulo anterior (XP). Se incluirá la elaboración de la Tarjeta de Historias de Usuarios, los requerimientos funcionales y no funcionales de la aplicación, Historia de Usuario Dividido en Tareas, Tareas de Ingeniería, Tarjetas de CRC (Clase, Responsabilidad y Colaboración), Patrones de Diseño así como la arquitectura usada.

Capitulo 3: Implementación y Pruebas del Sistema.

Se analizan el Sistema de Archivos, Estándares de Implementación, Pruebas de Unidad por Clases JUnit, el uso de los patrones de arquitectura y métricas de diseño. Se realizan pruebas de unidad y de interfaz para comprobar que se cumple con el objetivo planeado en cuanto a su funcionamiento, y desarrollar un caso de estudio para validar la herramienta.

Capitulo 1: Fundamentación Teórica.

Introducción.

Este capítulo tiene como objetivo introducir los principales conceptos y términos empleados en la Gestión de Configuración de Software, la Gestión de Cambios y el procedimiento utilizado por la facultad 3, también se describen las posibles tecnologías y metodologías que podrán ser utilizadas para la solución del problema planteado.

Gestión de Configuración

A continuación se muestran diferentes definiciones brindadas por distintos autores sobre Gestión de Configuración de Software (GCS).

Angélica de Antonio define por Gestión de Configuración a: "… disciplina, cuya misión es controlar la evolución de un sistema software" (Antonio, 2001).

Babich plantea que "Al arte de coordinar el desarrollo de software para minimizar la confusión se denomina Gestión de Configuración. La Gestión de Configuración es el arte de identificar, organizar y controlar las modificaciones que sufre el software que construye un equipo de programación. El objetivo es maximizar la productividad minimizando los errores" (Babich, 1986).

Para Rational la GCS "describe la estructura del producto e identifica los elementos que lo constituyen y que son tratados como entidades que pueden ser puestas bajo control de versiones en el proceso de GCS. La GCS tiene que ver con la definición de la configuración así como la construcción, el etiquetado y recolección de versiones de los artefactos" (Rational, 2003).

Según Brown la Gestión de Configuración es la que administra y controla el contenido, el cambio o el estado de la información compartida en un proyecto.
Su propósito fundamental es establecer y mantener la integridad y el control en los productos de software a lo largo del ciclo de vida del proyecto (Norm, et al., 1998).

La definición que brinda IEEE sobre GCS plantea que "Gestión de Configuración es la disciplina que abarca todo el ciclo de vida de la producción de software y productos asociados. Específicamente, requiere de la identificación de los componentes a

controlar y la estructura del producto, controla todos los cambios sobre los elementos y garantiza mecanismos para auditar todas las acciones." (Appleton, 2000) (IEEE, 1987) (IEEE, 1990).

Refiriéndose al tema Pressman plantea que la Gestión de configuración es el arte de identificar, organizar, y controlar las modificaciones que sufre el software que construye un equipo de programación. La meta es maximizar la productividad minimizando los errores (Pressman, 2000).

Otra definición es la que propone la Norma ISO 9000-3:1991, donde se establece que la Gestión de Configuración de Software provee mecanismo para identificar, controlar y dar seguimiento a cada una de las versiones de los elementos que conforman al producto software (Bamford, 1995).

En el libro "Software Reuse: Architecture, Process, and Organizartions for Busisness Success" se plantea que la Gestión de Configuración es: Proceso de soporte cuyo propósito es identificar, definir y almacenar en una línea base los elementos de software; controla los cambios, reporta y registra el estado de los elementos y de las solicitudes de cambio; asegura la completitud, consistencia y corrección de los elementos; controla, almacena, maneja y libera los elementos asociados al producto de software" (Jacobson, 1997).

Una vez revisadas en la bibliografía estas definiciones se considera la más completa la de Ivar Jacobson, Martín Griss y Patrick Jonson en el libro "Software Reuse: Architecture, Process, and Organizartions for Busisness Success", la más empleada es la que plantea IEEE aunque se considera incompleta ya que esta no incluye el control de procesos, ni el control de esfuerzo de los desarrolladores.

Para la Gestión de Configuración del Software, un Elemento de Configuración del Software es "toda la información o productos utilizados o producidos en un proyecto como resultado del proceso de Ingeniería de Software" (Antonio, 2001).

Un Elemento de Configuración de Software (ECS) es todo lo que sea puesto bajo control por las actividades de Gestión de Configuración del Software, como pueden ser:

- La especificación del sistema.

- El plan del proyecto.
- La especificación de requisitos.
- Un prototipo, ejecutable o en papel.
- El diseño preliminar.
- El diseño detallado.
- El código fuente.
- Programas ejecutables.
- El manual de usuario.
- El manual de operación e instalación.
- El plan de pruebas.
- Los casos de prueba ejecutados y los resultados registrados.
- Los estándares y procedimientos de ingeniería de software utilizados.
- Los informes de problemas.
- Las peticiones de mantenimiento.
- Los productos hardware y software utilizados durante el desarrollo.
- La documentación y manuales de los productos hardware y software utilizados durante el desarrollo.
- Diseños de bases de datos.
- Contenidos de bases de datos.

Para cada proyecto concreto se debe determinar qué se va a considerar como ECS. Cada uno debe constituir un elemento completo que se pueda controlar por separado.

Como ha quedado claro hasta el momento la Gestión de Configuración forma parte de las disciplinas de control en el desarrollo de software, para que esta disciplina se cumpla con calidad se han definido las siguientes actividades:

- Identificación de la Configuración: Consiste en identificar la estructura del producto, sus componentes y el tipo de estos, y en hacerlos únicos y accesibles de alguna forma.
- Gestión de Cambios en la Configuración: Consiste en controlar las versiones y entregas de un producto y los cambios que se producen en él a lo largo del ciclo de vida.
- Generación de Informes de Estado: Consiste en informar acerca del estado de los componentes de un producto y de las solicitudes de cambio, recogiendo estadísticas acerca de la evolución del producto.

- Auditoría de la Configuración: Consiste en validar la completitud de un producto y la consistencia entre sus componentes, asegurando que el producto es lo que el usuario quiere. (Antonio, 2001)

El Proceso Unificado de Rational (RUP) de IBM es el resultado de varios años de desarrollo y uso práctico en el que se han unificado técnicas de desarrollo a través del UML, ha logrado convertirse en la metodología estándar más utilizada. La metodología RUP propone o estandariza los artefactos generados en cada flujo de trabajo aún cuando se trate de proyectos de pequeñas dimensiones, además RUP propone roles específicamente diseñados para encargarse de la Gestión de la Configuración como el Gestor de Configuración y el Gestor de Cambios.

En la *figura 1.1* se muestran las actividades que propone RUP en el Flujo de trabajo que define como Gestión de Configuración y Cambios, así como los trabajadores que responden por ellas:

Ilustración 1: Actividades del Flujo de Trabajo: Gestión de Configuración y Cambios (Rational, 2003)

Gestión de Cambios.

El control de los cambios constituye la actividad más importante dentro de la Gestión de Configuración del Software, su objetivo fundamental es establecer un mecanismo para controlar los cambios a los elementos de configuración, por el cual regirse rigurosamente, teniendo en cuenta que el cambio puede ocurrir en cualquier momento durante el desarrollo del software. Un producto de software es intangible y por lo general muy abstracto, esto dificulta la definición del producto y sus requisitos, sobre todo cuando no se tiene precedentes en productos de software similares. Esto hace que los requisitos sean difíciles de consolidar tempranamente. Así, los cambios en los requisitos son inevitables, no sólo después de entregado el producto sino también durante el proceso de desarrollo (3.0, 2006).

Pressman refiriéndose a este tema de los cambios en el desarrollo del software expresa: “Cuando se construye software de computadora, los cambios son inevitables, lo que puede en algún momento determinado ocasionar confusión. La confusión surge cuando no se han analizado los cambios antes de realizarlos, no se han registrados antes de implementarlos, no se les ha comunicado a aquellas personas que necesitan saberlo o no se han controlado de manera que mejoren la calidad y reduzcan los errores” (Pressman, 2002).

Debido a que el sistema cambiará sin importar el momento del ciclo de vida en el cual se encuentre, la Gestión de Cambios constituye la actividad más importante de la Gestión de Configuración y esta tiene como objetivo principal proporcionar un mecanismo para controlar los cambios. Es necesario que el equipo de desarrollo esté preparado para enfrentar el cambio de manera adecuada en el momento que sea necesario, aunque muchas veces las solicitudes de cambio se producen durante la fase de mantenimiento del producto. La base del éxito al enfrentarse a un cambio está en saber equilibrar la forma en que este se implementa ya que un uso excesivo puede llegar a convertirlo en una pesada carga que en vez de resolver, cree problemas. A simple vista la Gestión de Cambios lejos de traer beneficios, podría llegar a entorpecer el proceso de desarrollo, por la carga adicional que supone el tener que disponer de personal que se encargue de estas actividades, así como la necesidad de adquirir nuevos conocimientos. Hasta el momento se han definido cuatro fuentes fundamentales de cambios, estas son:

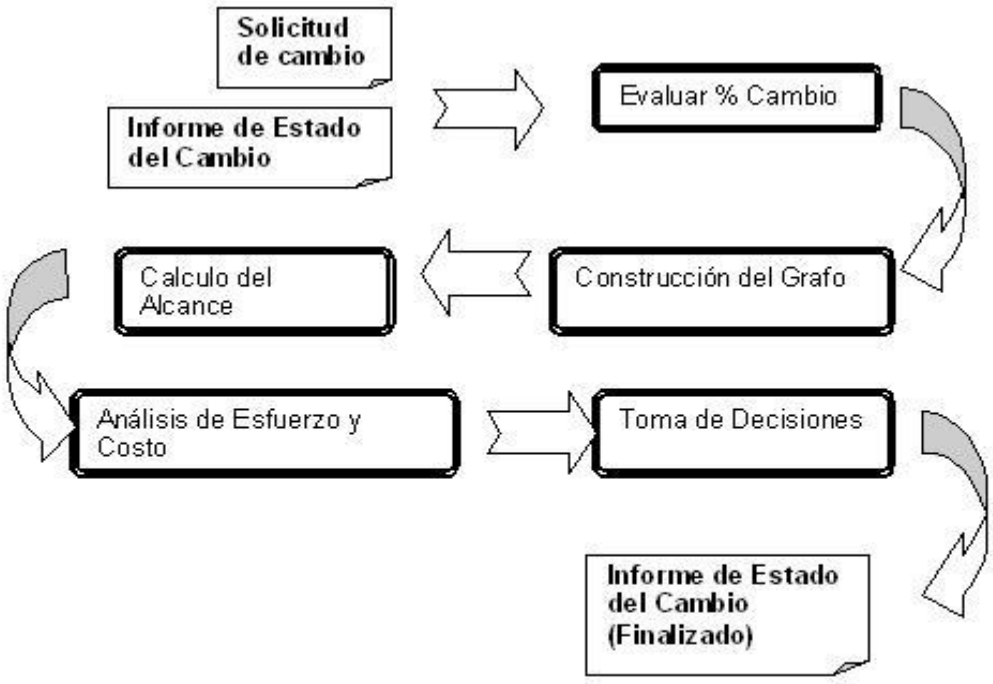
- Nuevos negocios o condiciones comerciales que dictan los cambios en los requisitos del producto o en las normas comerciales.

- Nuevas necesidades del cliente que demandan la modificación de los datos producidos por sistemas de información, funcionalidades entregadas por productos o servicios entregados por un sistema basado en computadora.
- Reorganización o crecimiento/reducción del negocio que provoca cambios en las prioridades del proyecto o en la estructura del equipo de ingeniería del software.
- Restricciones presupuestarias o de planificación que provocan una redefinición del sistema o producto (Pressman, 2000).

Procedimiento de Gestión de Cambios a Utilizar

Este es un procedimiento elaborado por dos estudiantes para su tesis de diplomado en el año 2008, dicho procedimiento es un híbrido entre varios procedimientos, quedó validado y recomendado para ser usado en la facultad 3.

Ilustración 2: Procedimiento para la Toma de decisiones en la Gestión de Cambios (Quintana Torres, et al., 2008)**.**



Desarrollo del Procedimiento

1. Evaluar % Cambio:

En esta actividad se establecen como entradas la Solicitud de Cambio y el Informe de Estado del Cambio, se realizará con el objetivo de evaluar el porciento que cambiará el ECS que se propone sea cambiado sin tener en cuenta su impacto sobre el resto de

los elementos. El responsable de esta actividad será la persona que creó el ECS que se propone cambiar. (Quintana Torres, et al., 2008)

2. Construcción del Grafo:

El objetivo de esta actividad es construir un grafo dirigido y ponderado donde los nodos representen los ECS que integran la Línea Base y las aristas representen las relaciones que existen entre ellos, en este caso se ponderarán los nodos y las aristas en dependencia de la clasificación que estos tengan. Es importante aclarar que el grafo debe construirse cuando se enfrente por primera vez un cambio a cualquier ECS que forme parte de esa línea base y a partir de este debe irse actualizando para los cambios posteriores. (Quintana Torres, et al., 2008)

Cálculo del Alcance

Una vez que el grafo este elaborado o actualizado, el Presidente del CCC prosigue a calcular el alcance del cambio, dentro de esta actividad se calcula además el porciento de cambio que sufrirá el proyecto y la complejidad promedio de los ECS que serán afectados por el cambio. Esta actividad depende completamente de la obtención de datos del grafo ya construido. Al concluir esta actividad se actualiza el documento Informe de Estado del Cambio colocando en este los valores arrojados en el cálculo. (Quintana Torres, et al., 2008)

3. Análisis de Costo

En esta actividad el planificador, basándose en los valores numéricos que arroje el cálculo del alcance y empleando los métodos de estimación existente el equipo deberá realizar un análisis del esfuerzo y costo asociado al cambio, debe reevaluar costo total del proyecto. Después de haber realizado las estimaciones deberá reflejarlas en el Documento de Informe de Estado del Cambio. (Quintana Torres, et al., 2008)

4. Toma de Decisiones.

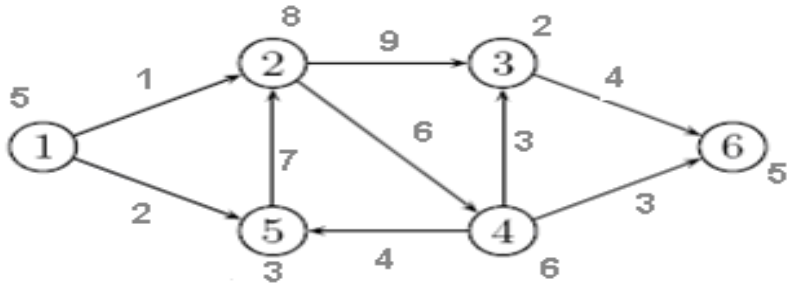
Por último y no por ello menos importante se procede a efectuar la toma de decisiones por parte del Comité de Control de Cambios mediante una reunión para adoptar una conducta con respecto al Cambio, donde se tendrán en cuenta los beneficios o prejuicios que tendrá para el proyecto. Al concluir esta reunión el Presidente del CCC deberá reflejar en el documento Informe de Estado del Cambio la decisión tomada y en caso de que el cambio sea rechazado explicar las causas. En epígrafes posteriores

se explicarán detalladamente los pasos a seguir para el cumplimiento de estas actividades. (Quintana Torres, et al., 2008)

Grafos Dirigidos y Ponderados.

En los Grafos Dirigidos las relaciones sobre los vértices son asimétricas y las aristas se representan mediante un par ordenado. Un Grafo ponderado o etiquetado es aquel en que cada arco, nodo, o ambos, tienen asociada una etiqueta o peso. En un Grafo se denomina camino a una sucesión de nodos adyacentes, la longitud de un camino es el número de arcos o aristas que lo conformen, un camino simple es en el que todos sus vértices, excepto, tal vez, el primero y el último, son distintos, en un grafo se llama bucle a un ciclo de longitud igual a uno, un grafo acíclico es aquel en el que no existen ciclos. Se denominan nodos adyacentes si entre ellos existe una arista. El grado de un nodo es el número de arcos que inciden en él. Entonces el grado de un grafo será el máximo grado de sus vértices. (Quintana Torres, et al., 2008)

Ilustración 3: Grafo Dirigido Ponderado



Recorridos.

Recorrer un grafo significa tratar de alcanzar todos los nodos que estén relacionados con uno que llamaremos nodo de salida. Existen básicamente dos técnicas para recorrer un grafo: el recorrido en anchura; y el recorrido en profundidad (Augenstein, et al., 1993).

- Recorrido (o búsqueda) en amplitud (breadth-first search): El recorrido en anchura supone recorrer el grafo, a partir de un nodo dado, en niveles, es decir, primero los que están a una distancia de un arco del nodo de salida, después los que están a dos arcos de distancia, y así sucesivamente hasta alcanzar todos los nodos a los que se pudiese llegar desde el nodo salida.
- Recorrido (o búsqueda) en profundidad (depth-first search): El recorrido en profundidad trata de buscar los caminos que parten desde el nodo de salida

hasta que ya no es posible avanzar más. Cuando ya no puede avanzarse más sobre el camino elegido, se vuelve atrás en busca de caminos alternativos, que no se estudiaron previamente. (Quintana Torres, et al., 2008)

Herramientas para la identificación de la configuración de software.

Se sabe que el proceso de gestión de cambios se encuentra incluido implícitamente dentro del proceso de gestión de Configuración del Software de ahí que empezaremos viendo como se encuentran en el mundo los productos software que lo automaticen. La mayoría de estos productos en general se encuentran dedicados al idioma inglés por lo que en español es casi imposible realizar una investigación de este tipo. Existe una inmensa gama de productos que generalmente se caracterizan por tratar y controlar:

- La elaboración de código fuente por varios desarrolladores simultáneamente.
- El seguimiento del estado de las versiones y sus cambios y la conducción de la integración de las partes del software en un solo producto de software.

Estas herramientas por muy bien elaboradas que estén nunca podrán darle una solución total al proceso pues a menudo se han identificado que no cumple requisitos técnicos tales como:

- Apoyo a diferentes plataformas.
- Iniciar el proceso de construcción.
- Conexión a los bancos de datos existentes.
- Integración a la organización existente.

Por esa razón ofrece una mayor flexibilidad una solución que integre herramientas parciales que sean más fáciles de integrar en el proceso existente.
Ahora, debido a la gran cantidad de productos existentes se van a mencionar los más importantes y usados a nivel mundial dividiendo estos en dos grupo, los de Software Libre y lo de Software Propietario, brindando una breve de cada uno de ellos

Software Propietario

PowerDesigner

La Herramienta Líder en Modelamiento Empresarial PowerDesigner desarrollada por SyBase, es número uno de la industria, permite a las empresas, de manera más fácil, visualizar, analizar y manipular metadatos, logrando un efectiva arquitectura empresarial de información.

PowerDesigner para Arquitectura Empresarial también brinda un enfoque basado en modelos, el cual permite alinear al negocio con la tecnología de información, facilitando la implementación de arquitecturas efectivas de información empresarial. Brinda potentes técnicas de análisis, diseño y gestión de metadatos a la empresa.

PowerDesigner combina varias técnicas estándar de modelamiento con herramientas líder de desarrollo, como .NET, SyBase WorkSpace, SyBase Powerbuilder, Java y Eclipse, para darle a las empresas soluciones de análisis de negocio y de diseño formal de base de datos. Además trabaja con más de 60 bases de datos relacionales.

Beneficios

Alinea el negocio y la tecnología de información para mejorar la productividad.
Brinda soporte abierto a ambientes heterogéneos de todas clases.
Es altamente personalizable, permitiendo acogerse a los estándares y regulaciones.
Facilita la arquitectura empresarial, documentando los sistemas existentes.
Aumenta la agilidad del negocio con "Link & Sync" y análisis de impacto.

El precio de mercado se encuentra entre $ 2995 a $ 7495 por desarrollador que lo utilice. (SyBase, 2008).

Visual SourceSafe

Visual SourceSafe de Microsoft es un sistema de control de versiones en el nivel de archivos, que permite a muchos tipos de organizaciones trabajar en distintas versiones de un proyecto al mismo tiempo. Esta funcionalidad es especialmente ventajosa en un entorno de desarrollo de software, donde se usa para mantener versiones de código paralelas. Sin embargo, el producto también se puede utilizar para mantener archivos en cualquier otro tipo de equipo.

Visual SourceSafe incluye, como mínimo, las siguientes funciones:

- Ayuda al equipo para evitar la pérdida accidental de archivos.
- Permite realizar un seguimiento de las versiones anteriores de un archivo.

- Admite la bifurcación, el uso compartido, la combinación y la administración de versiones de archivos. Realiza el seguimiento de las versiones de proyectos completos.
- Realiza el seguimiento del código modular (un archivo que se reutiliza, o se comparte, en varios proyectos).

El precio de mercado se encuentra entre $65.00 a $125.00. (Microsoft, 2009)

Perforce

Fundada en 1995, Perforce software ha mantenido una singular atención a la administración de la configuración del software (SCM), que ha permitido que el sistema de SCM Perforces se convierta en una de las mejores soluciones probadas.

Perforce se basa en un modelo centrado en los datos, mediante el uso de una base de datos de metadatos, Perforce Servidor realiza un seguimiento de lo que los archivos que usted (y otros usuarios) están trabajando en todo momento, la eliminación de tráfico de red innecesario.

Posee una gran cantidad de plataformas con interfaces gráficas, los usuarios remotos experimentarán los tiempos de respuesta rápidos y tienen acceso en tiempo real a la actividad de proyecto y la información de estado. Una arquitectura abierta compatible con la integración con cualquier IDE y una amplia variedad de aplicaciones de herramientas de gestión de ciclo de vida. Fácil Administración. Flexible, rápido, intuitivo y la capacidad de la Subdivisión y fusión.

Precio de mercado alrededor de $800 por usuario. (Perforce, 2008)

Software libre

CVS (Concurrent Versions System)

CVS es muy popular, cuando se publicó el CVS, fue una nueva e importante innovación en software de gestión de la configuración. Sin embargo, el CVS muestra una serie de limitaciones que producen incomodidad, los cambios son seguidos por el archivo en lugar de por el cambio, renombrar ficheros y directorios es torpe. Algunos de los mantenedores del original CVS han declarado que el código de CVS se ha vuelto demasiado crujiente a mantener de manera eficaz. Estos problemas llevaron a los principales desarrolladores de CVS para empezar de nuevo y crear Subversion. (CVS, 2008)

Subversion

Subversion (SVN) es un nuevo sistema, con la intención de ser un simple reemplazo de CVS. Subversion 1.0, lanzado 24 de febrero de 2004 el cual posee varias deficiencias.

Es software libre bajo una licencia de tipo Apache/BSD y se le conoce también como svn por ser ese el nombre de la herramienta de línea de comandos. Una característica importante de Subversion es que, a diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo.

Carencias:

El manejo de cambio de nombres de archivos no es completo. Lo maneja como la suma de una operación de copia y una de borrado.

No resuelve el problema de aplicar repetidamente parches entre ramas, no facilita el llevar la cuenta de qué cambios se han trasladado. Esto se resuelve siendo cuidadoso con los mensajes de commit. (Collins-Sussman, et al., 2002-2008)

GNU Arch

GNU Arch es una revisión del sistema de control, similar en propósito a herramientas tales como CVS, SCCS, y Subversion. Se utiliza para realizar un seguimiento de los cambios realizados en un árbol de fuentes y para ayudar a los programadores combinar y manipular los cambios realizados por varias personas o en diferentes momentos.

Tiene algunas características que lo hacen especialmente útil para los proyectos de software libre: es fácil de aprender, es barato y fácil de administrar, es un sistema distribuido por lo que no hay necesidad de dar permisos de escritura a cada participante en el proyecto, tiene un apoyo excelente para los tipos de ramificación que pueden hacer que un equipo de proyecto de software libre brille. (GNU, 2008)

Bazaar-ng

Bazaar-ng es un sistema de control de versiones distribuido que, al igual que CVS o Subversion, nos permite guardar progresivamente los cambios que vayamos realizando sobre un conjunto de archivos de texto (habitualmente código fuente), recuperar versiones anteriores, mostrar diferencias, integrar el trabajo de diversos programadores, etc.…

Sin embargo, a diferencia de CVS o Subversion, Bazaar-ng nos permite trabajar de formas mucho más flexibles… desde el típico esquema cliente-servidor hasta la descentralización de los repositorios. (Marblestation, 2006)

Monotone

Monotone es una herramienta software de fuente abierta para control distribuido de versiones.

Monotone registra revisiones de ficheros, agrupa conjuntos de revisiones ('changesets') y mantiene el histórico tras cambios de nombre. El principio de diseño es el de operación distribuida haciendo gran uso de primitivas criptográficas para trazar revisiones de ficheros (mediante el hash seguro SHA1) y para autenticar acciones de usuarios (mediante firmas criptográficas RSA). Cada participante mantiene su propio almacén de revisiones históricas en una base de datos SQLite local.

Monotone es especialmente potente en su soporte de la rutina de separación/integración ('diverge/merge'), que consigue en parte al permitir siempre hacer 'commit' antes de integrar. Las revisiones se intercambian haciendo uso del protocolo propio netsync, que tiene su propio puerto (4691) asignado por IANA y dispone de un plugin para Wireshark que permite el análisis de tráfico (netsync comparte cierto terreno conceptual con rsync y cvsup). El enfoque del proyecto está en la integridad por encima del rendimiento.

Antes de ciertas optimizaciones importantes en la revisión 0.27, se solía culpar a este énfasis en la corrección en lugar de optimización como la causa de malas experiencias iníciales. La primera acción de un usuario nuevo suele ser sincronizar (clonar) una gran base de datos de Monotone ya existente, acción que solía llevar horas de ejecución debido a las comprobaciones exhaustivas de validación e integración que realiza Monotone cuando se mueven revisiones por la red. Una vez poblada la base de datos inicial (clon), las acciones posteriores suelen ser rápidas. Sigue habiendo espacio para optimizaciones en algunas de las funciones menos habituales. (Monotone, 2008)

Herramientas para llevar a cabo toma de decisiones en la Gestión de Cambios.

En el mundo existe una gran cantidad de productos que brinda esta funcionalidad y se agrupan en el tipo de control de revisiones también conocidos como control de versiones. Son más utilizados en la ingeniería y desarrollo de software para la gestión en curso de desarrollo de documentos digitales, código fuente, planos o modelos electrónicos, y otros elementos que puedan ser trabajados por un equipo de personas. Los cambios en estos documentos se identifican generalmente por un creciente número de asociados o código de letras, el nombre de "número de revisión", "nivel de revisión" o, simplemente, "revisión" y asociada históricamente con la persona que efectúa el cambio y la fecha en que se realiza.

A continuación se presenta una serie de productos que como anteriormente se presentó los dividiremos en software libre y propietario, además de dividirlos en enfoque distribuido y modelo cliente-servidor.

Enfoque distribuido.

En el enfoque distribuido, cada desarrollador trabaja directamente con su propio repositorio local, y los cambios son compartidos entre los repositorios como un paso aparte.

Software Propietario

Code Co-op

Code Co-op es el peer-to-peer sistema de control de versiones realizadas por Software Confiable.

Características distintivas.

Code Co-op es un sistema de control de versiones distribuido de tipo cerrado.

Utiliza peer-to-peer para compartir proyectos de arquitectura entre los desarrolladores y para controlar los cambios a los archivos. En lugar de utilizar una base de datos centralizada (el repositorio), que reproduce su propia base de datos en cada equipo que participa en el proyecto.

Las réplicas e interacciones son sincronizadas por el intercambio. El intercambio de secuencias de comandos puede realizarse mediante diferentes transportes, incluido el correo electrónico (soporte para SMTP y POP3, integración con clientes MAPI, Gmail) y LAN.

Code Co-op ha incorporado en Peer-to-peer "wiki sistema que puede ser usado para integrar la documentación con un proyecto de software. También es posible crear texto basados en Wiki bases de datos.

Características estándar

Distribuido desarrollo a través de E-mail, LAN o VPN.

Cambio de modelo basado en las modificaciones de varios archivos como se comprueba en una transacción.

Adiciones, supresiones, renombramientos, movimientos de Archivos se tratan en el mismo nivel como las ediciones de estos, se pueden añadir en cualquier combinación de revisiones de un control de cambios.

El archivo de los cambios puede ser revisado antes de un control en el uso de un built-in o definidos por el usuario.

Sincronización de cambios pueden ser revisados de la misma manera por los beneficiarios.

Integración con Microsoft SCC clientes, incluidos los de Visual Studio.

El historial del proyecto es replicado en cada máquina, esta puede ser revisada, comparada o restaurada.

Precio de mercado $149 a $199 (Reliable, 2008)

Plastic SCM

Plastic SCM es un sistema desarrollado por Codice Software. Plastic SCM intenta centrarse en el desarrollo paralelo, ramificando y fusionando la seguridad.

Plastic es un sistema innovador para la Gestión de la Configuración de Software (SCM). El usuario no tiene que trabajar en los ficheros desde el repositorio sino que utiliza un programa cliente para gestionar su “espacio de trabajo”, este es una zona en su ordenador que contiene una copia de una parte del repositorio.

Plastic SCM introduce diferentes aspectos novedosos tanto en su diseño como en la manera en la que permite manejar la información, permite el desarrollo paralelo lo cual aporta una mayor cooperación entre el equipo de trabajo; así como control total sobre los cambios realizados por cada desarrollador y cada integración y nuevas técnicas de visualización que ofrecen características de alto nivel para empresas de cualquier tamaño, y por supuesto, también para equipos distribuidos. (Codice, 2007)(Ver Anexo 2)

BitKeeper

BitKeeper es el sistema de gestión de código fuente usado hasta ahora para el kernel de Linux. Aunque no es software libre, Linux comenzó a usarlo hace unos años porque no había ninguna alternativa libre que proporcionase las funcionalidades necesarias. (Bitkeeper, 2008)

Precio (variable).

Software Libre

GIT

El diseño fue inspirado en Bitkeeper Monotone.Git y fue originalmente diseñado con un bajo nivel de control de versiones del sistema motor, en la parte superior del que podría escribir otras interfaces, tales como Cogito o StGIT. Sin embargo, el núcleo del proyecto Git se ha convertido en una revisión completa del sistema de control que se puede utilizar directamente

El diseño es una síntesis de la experiencia de Torvalds en el mantenimiento de un gran proyecto de desarrollo distribuido, su profundo conocimiento de los resultados del sistema de archivos, y una necesidad urgente de elaborar un sistema de trabajo en el corto plazo. Estas influencias dieron lugar a la aplicación con las siguientes opciones:

- Un fuerte apoyo para el desarrollo no lineal. Git apoya la rápida fusión de sucursales, e incluye herramientas específicas para la visualización y la navegación no lineal de la historia del desarrollo. Un supuesto básico en Git es un cambio que se fusionará con más frecuencia de lo que está escrito, ya que se pasa en torno a diversos encuestados.

- Desarrollo distribuido. Al igual que Darcs, Bitkeeper, Mercurial, ESK, Bazar y Monotone, Git le da a cada desarrollador una copia local de todo el desarrollo de la historia, y los cambios se copian desde un depósito a otro. Estos cambios se importan como el desarrollo adicional de las ramas, y pueden combinarse de la misma manera que una rama de desarrollados localmente.

- Repositorios se pueden publicar a través de HTTP, FTP, rsync, o un protocolo Git más bien un simple enchufe o ssh. Git también tiene un servidor CVS emulación, que permite la utilización de los actuales clientes y CVS IDE plugins para acceder a repositorios Git.

- Autenticación criptográfica de la historia. Git La historia está almacenada de tal manera que el nombre de una revisión particular (un "comprometerse" en términos Git) depende del desarrollo completo de la historia que condujo a que se comprometan. Una vez que se publica, no es posible cambiar las versiones antiguas sin que se note. La estructura es similar a la de un árbol, pero con datos adicionales a los nodos, así como las hojas. (Mercurial y Monotone también tienen esta propiedad.)

- Conjunto de herramientas de diseño. Git fue diseñado como un conjunto de programas escritos en C, y una serie de scripts de shell que proporcionan envoltorios alrededor de los mismos. Aunque la mayoría de los guiones se han reescrito en C como parte de un esfuerzo constante por el puerto a Microsoft Windows, el diseño sigue siendo, y es fácil de la cadena de los componentes juntos para hacer otras cosas inteligentes. (GIT, 2008)

CodeVille

Codeville es un sistema de control de versiones distribuidas. Comenzó con una idea innovadora para un algoritmo de combinación y ha crecido a partir de ahí. Está diseñado para ser fácil de usar y la escala de pequeños proyectos personales a los grandes distribuido. (Codeville, 2008)

Si desea saber por qué hay una necesidad de nuevos algoritmos de fusión, lo que consideran el principal desarrollador monótono tiene que decir:

- Situación actual.
- Simple y robusto algoritmo de combinación de probada eficacia.
- Se destaca la fusión de las ramas y sin restricciones.
- Interfaz de usuario sencilla.
- Casi trivial, utilizar para sus propios proyectos sin ejecutar un servidor.
- Mantiene modelo cliente / servidor para aquellos no familiarizados con los sistemas distribuidos
- Transparente, se extiende el modelo cliente / servidor distribuido al caso, todos los depósitos están contemplados en el URL.
- Simplificación de la fusión de sucursales y de CVS, que "simplemente funciona"
- Conjunto básico de comandos con funcionalidad rica.

- Nombre de archivo comodín'...' puede utilizarse en lugar de con el futzing unix 'encontrar' comando.
- Buenos resultados, incluso en los grandes proyectos.
- Autenticación fuerte.
- SRP autenticación significa que no exponga su contraseña para servidores remotos por accidente.
- HMAC códigos de protección contra la conexión de secuestro
  agente para que no tenga que escribir su contraseña todo el tiempo.
- Desconectado operacionalmente.
- Historia de la navegación y diffing.
- Potente diffing comandos de navegación y la historia.
- Inmutable y verificable de la historia.
- Amplia utilización de hashes incluyendo la totalidad de los siguientes.
- Identificadores internos de los archivos y directorios.
- Archivo de los deltas.
- Cambiar números.
- Renombrar archivos y directorios correctamente que se ocupa de todas (y hay muchos) casos esquina y los conflictos.
- Cruz plataforma, se ejecuta en todas las principales plataformas.
- Liberado como código abierto.

Mercurial

Mercurial es una cruz-plataforma, herramienta de control distribuido de revisión para los desarrolladores de software. Es aplicado principalmente utilizando el lenguaje de programación Python, pero incluye una aplicación binaria diferencia por escrito en C. Mercurial fue escrito para funcionar en Linux. Se ha portado a Windows, Mac OS X, y la mayoría de los otros sistemas Unix. Mercurial es principalmente un programa de línea de comando.

Todas las operaciones de Mercurial se invocan como palabra clave para las opciones de su programa controlador HG, una referencia al símbolo químico el elemento mercurio.

Mercurial los principales objetivos de diseño son de alto rendimiento y escalabilidad, descentralizado, plenamente distribuidos de desarrollo colaborativo, sólido manejo de texto y archivos binarios, y avanzadas capacidades de la fusión de las ramas y, sin dejar de ser conceptualmente simples. Se incluye una interfaz web integrada.

El creador y desarrollador principal de Mercurial es Matt Mackall. El código fuente está disponible bajo los términos de la GNU General Public License versión 2, que reúnen las Mercurial como el software libre.

Información Técnica

Mercurial utiliza hash para identificar revisiones. Para acceso al repositorio a través de una red, usos basados en HTTP protocolo que pretende reducir las peticiones de ida y vuelta, nuevas conexiones y los datos transferidos. (Selenic, 2008)

Fossil

Fósiles es un software de control de versiones distribuido que incluye un sistema integrado distribuido un wiki y un distribuido sistema de seguimiento de fallos en un solo, de fácil uso, independiente ejecutable. Fósiles es libre desde 2007-07-21 alojamientos en dos servidores. Puede descargar la última fuente y compilarlo usted mismo. O usted puede agarrar los binarios pre-compilados. (Fossil, 2008)

Resumen de características:

- Flexible del flujo de trabajo:
  - Desconectado, desarrollo distribuido como GIT, monótono, mercurial, y Bitkeeper.
  - O, el modelo cliente / servidor de operación como CVS y Subversion,
  - O, las operaciones en los depósitos locales,
  - O, todo lo anterior, al mismo tiempo.
- Integrado, distribuido de seguimiento de fallos y distribuido wiki.
- Incorporada en la interfaz web que soporta las excavaciones arqueológicas de profundidad a través de la historia del proyecto.
- Todas las comunicaciones en red a través de HTTP proxy con el apoyo para que todo funcione desde detrás de cortafuegos restrictivos.
- Todo (cliente, servidor y servicios) se incluye en un solo auto-ejecutable - trivial instalar.
- Servidor se ejecuta como CGI, utilizando inetd / xinetd o utilizando sus propios incorporada, independiente del servidor web.
- La totalidad de un único proyecto que figura en el archivo de disco (una base de datos SQLite.)

- Utiliza un formato de archivo permanente que está diseñado para ser leído, de búsqueda, y extensible por personas aún no nacidas.
- Auto-control de cambios del repositorio, es extremadamente poco probable que los datos nunca se pierdan a causa de un error de software.
- Ridículamente fácil de instalar y operar.
- Licencia: GPL.

Enfoque Cliente-Servidor.

En el modelo cliente-servidor, los desarrolladores utilizan un único repositorio compartido.

Software Propietario

IBM Rational ClearCase

IBM Rational ClearCase proporciona una gestión del ciclo de vida y control de los activos de desarrollo de software. Con un control integrado de versiones, una gestión del espacio de trabajo automatizado, un soporte de desarrollo en paralelo, una gestión de línea base y gestión de builds y releases, Rational ClearCase proporciona las funciones necesarias para crear, actualizar, crear, ofrecer, reutilizar y mantener los activos más importantes del negocio.

Rational ClearCase puede ayudarle a aumentar la productividad a través del desarrollo en paralelo, obtener uno tiempo más reducido del ciclo de build/release y una mayor reutilización de software. La integración con los IDE líder, incluido Rational Application Developer, WebSphere Studio, Microsoft Visual Studio .NET y la infraestructura Eclipse de código abierto agiliza aún más el desarrollo.

Unas interfaces locales, remotas y web permiten un acceso prácticamente a todos sitios y en cualquier momento.

Un soporte de desarrollo de Linux, Windows, Unix y mainframe (z/OS) permite un desarrollo de builds y de aplicaciones a nivel empresarial.

Rational ClearCase se integra sin fisuras con Rational ClearQuest para obtener una solución completa de gestión de la configuración del software. (IBM, 2008)

Star Team, Borland

Proporciona una solución completa para la gestión de los cambios y la configuración que incluye la integración con la gestión de requerimientos, tracking de defectos, versiones de código fuente y seguimiento de proyectos y tareas. Incluye un workflows Customizable que permite unificar equipos dentro de un ambiente centralizado y su

facilidad de empleo y su arquitectura eficaz reducen los costos en forma apreciable. Resumidamente Borland StarTeam ofrece una gama completa de capacidades para la gerencia de los cambios y la configuración de software en una plataforma integrada que apoye a equipos del desarrollo de todos los tamaños. (Borland, 2008)

Software Libre

CVSNT

CVSNT es un software que se utiliza para llevar un seguimiento de los cambios a los archivos almacenados en un ordenador. Esta es la función en el corazón de todo el código fuente de Gestión, Gestión de documentos y de Sistemas de Gestión de Configuración. CVSNT profesional y CVS Suite incluye herramientas adicionales para ayudar en el despliegue de los archivos de prueba y los entornos de producción, o lo que el seguimiento de los procesos iniciados necesite los cambios y mucho más.

Después de analizar y estudiar profundamente con la investigación antes presentada se percata que un producto software que brinde la funcionalidad de toma de decisiones en el proceso de Gestión de Cambios no se encuentra todavía en el mercado. (CVSNT, 2008)

Tecnologías y Herramientas propuestas para el desarrollo del sistema.

Los lenguajes de programación y los IDE de desarrollo tienen suma importancia para cualquier proyecto de software. Pues con estos se han creado innumerables programas y herramientas que han ayudado al hombre a controlar de manera más sencilla a los ordenadores, así como de realizar múltiples tareas y actividades. Actualmente en el mundo existe una gran variedad de lenguajes y plataformas de desarrollo para aplicaciones de escritorio. En los últimos años los lenguajes y plataformas de desarrollo han ido evolucionando siempre con el objetivo de ofrecer una interfaz más amigable para el programador y abstraerlo de las funciones más elementales por lo que vamos a mostrar una breve reseña.

Lenguajes de Programación Java y C#.

Java:

Java es un lenguaje de programación creado por SUN Microsystems muy parecido al estilo de programación del lenguaje “C” y basado en el paradigma de programación orientada a objeto.

Entre las principales características de Java se encuentran:

- Sintaxis similar a la de C++ y C#. Aunque se simplifican algunas características del lenguaje como:
- La sobrecarga de operadores, el paso por referencia de parámetros, la gestión de punteros, la liberación de memoria y las instrucciones de pre compilación.
- Soporte homogéneo a la Programación Orientada a Objetos. A diferencia de C++, que puede considerarse un lenguaje multiparadigma, Java está diseñado específicamente para utilizar el paradigma de orientación a objetos.
- Independencia de la plataforma. En Java se pretende que con una sola compilación se obtenga código ejecutable en diferentes Sistemas Operativos e incluso sobre diferente hardware. (Javahispano, 2009)

C#:

C# es el nuevo lenguaje de propósito general diseñado por Microsoft para su plataforma .NET. Sus principales creadores son Scott Wiltamuth y Anders Hejlsberg, éste último también conocido por haber sido el diseñador del lenguaje Turbo Pascal y la herramienta RAD Delphi.

Este lenguaje posee una estructuración y una sintaxis muy parecida a la de C++ o Java, esto con el interés de Microsoft de atraer a los programadores de Java y C++, tomando las mejores características de estos lenguajes. También es un lenguaje orientado a objetos simple, elegante y con seguridad en el tratamiento de tipos, que permite a los programadores de aplicaciones empresariales crear una gran variedad de aplicaciones.

Entre las principales características de C Sharp se encuentran:

- Proporciona la capacidad de generar componentes de sistema duraderos debido a que posee total compatibilidad entre COM y plataforma para integración de código existente.
- Presenta gran robustez, gracias a la recolección de elementos no utilizados a esto se le conoce como liberación de memoria.
- Seguridad implementada por medio de mecanismos de confianza intrínsecos del código, presenta plena compatibilidad con conceptos de meta datos extensibles.

Para llevar a cabo el desarrollo de la herramienta que es objeto de esta investigación se decidió el uso del lenguaje de programación Java por las ventajas y potencialidades

que tiene frente a otros lenguajes, al poder implementarse con independencia de la plataforma y como software libre, permitiendo obtener productos de excelente calidad, en menor tiempo y, por consiguiente, con menores costos.

Java se distingue de otros lenguajes, en que es una plataforma completa de desarrollo, consta de un gran conjunto de componentes que se pueden reutilizar y mecanismos para extenderlos, facilitando la vida a los desarrolladores. Aunque al mismo tiempo obliga a tener buenas prácticas y buenos patrones de diseño a diversos problemas recurrentes de desarrollo. (Microsoft, 2009)

Entornos de Desarrollo Integrado para Java.

Eclipse IDE para Java:

Eclipse es un Entorno de Desarrollo Integrado (IDE) que en un principio fue diseñado y desarrollado por IBM y que luego fue lanzada a la comunidad de software libre, y que se distribuye mediante una licencia de código abierto, la Eclipse Public License (EPL). La plataforma de Eclipse integra herramientas de desarrollo, con una arquitectura abierta y basada en plug-ins. Además, da soporte a todo tipo de proyectos que abarcan desde el ciclo de vida del desarrollo de aplicaciones, incluyendo soporte para modelado.

La arquitectura de plug-ins permite integrar diversos lenguajes sobre un mismo IDE e introducir otras aplicaciones accesorias. Conservan el registro de las versiones, generan y mantienen la documentación de cada etapa del proyecto.

Como principales características Eclipse posee un editor visual con sintaxis coloreada, además de modificar e inspeccionar valores de variables. También avisa de los errores cometidos mediante una ventana secundaria, y depura el código que resida en una máquina remota. Eclipse permite agrupar código escrito y mostrado visualmente como una estructura empaquetada para que sea fácil poder seguir un código escrito. (Bermejo Sanz, et al., 2008)

En los últimos tiempos, la popularidad de Eclipse ha subido enormemente entre los desarrolladores no sólo de Java, sino también de los otros lenguajes a los que Eclipse da soporte, pues es una plataforma de herramientas universal y portable que proporciona un marco de trabajo o framework para desarrollar aplicaciones y herramientas.

NetBeans IDE para Java:

Netbeans es un IDE libre con ambiente integrado para desarrolladores de código abierto. Contiene todas las herramientas necesarias para el desarrollo de aplicaciones profesionales de escritorio, web, y de móviles en los lenguajes de programación java C, C++ y en Ruby. Este IDE puede ser ejecutado en distintas plataformas incluyendo Windows, Linux, Mac OS X y Solaris. Es muy fácil de instalar y permite una agradable interacción con el desarrollador.

Características principales

- Herramienta para construcción de interfaces de visuales (GUI)
- Posee una barra de herramientas que permite con facilidad la creación de interfaces, donde el desarrollador puede arrastrar diferentes componentes, alinearlos y programarlos.

Ventajas:

- Incorpora la línea de tecnología de Sun dentro del entorno.
- No hay que buscar plug-ins por todas partes, por lo regular todo se encuentra en la misma caja.
- La estructura de los proyectos está basada en ant, por lo tanto es muy personalizable, por si después de empaquetar un proyecto se quiere enviar en ftp, se puede hacer fácilmente con ant.
- Developer Collaboration: Es un plug-in que como su nombre lo indica, permite hacer desarrollos en equipo. Se puede modificar el mismo archivo a la vez y ver los cambios en tiempo real, se puede estar chateando o enviar pedazos de códigos.

Herramienta del NetBeans para el Modelado UML

Contiene una barra de herramientas para el modelado UML de la aplicación que se esté desarrollando. Desarrollo en C y C++ Está concebido para el desarrollo en Java, pero permite la potencialidad de que el desarrollador pueda programar en otros lenguajes como C y C++, pues contiene un editor, y herramientas de corrección del código para estos lenguajes (NetBeans, 2008)

Librerías y Frameworks para Java.

Org-netbeans-api-visual

El api proporciona un conjunto de piezas reutilizables-(witdges), que en su composición crean una visualización. Cada uno de estos tiene diferentes propiedades incluyendo el diseño y acciones asignadas. La biblioteca contiene witges predefinidos que pueden ser utilizados. Todos los elementos bases han sido declarados como interfaces o clases abstractas con el objetivo de su personalización a cada proyecto: WidgetAction, Anchor, AnchorShape, PointShape, Animator, Border, GraphLayout, LookFeel, Layout, SceneLayout, Router, CollisionsCollector.
Además de traer su implementación por defecto. (NetBeans, 2008)

Tabla 1: Org-netbeans-api-visual

org.netbeans.api.visual.action	Este paquete contiene ActionFactory clase de fábrica en que se construyen todas las widget-acciones previstas por la biblioteca.
org.netbeans.api.visual.anchor	Este paquete contiene Anchor interfaz que es utilizado por ConnectionWidget para definir su punto de origen y de destino.
org.netbeans.api.visual.animator	Este paquete contiene SceneAnimator clases que se utiliza para controlar las animaciones en una escena.
org.netbeans.api.visual.border	Este paquete contiene Border de interfaz que se utiliza para el suministro de una frontera de gráficos para un widget.
org.netbeans.api.visual.export	Este paquete contiene las clases necesarias para exportar una Scene a un archivo de imagen y obtener las coordenadas de la imagen exportada en HTML para crear un mapa de imagenan HTML image map.
org.netbeans.api.visual.graph	Este paquete contiene el gráfico incorporado en los modelos orientados.
org.netbeans.api.visual.graph.layout	Este paquete contiene incorporados en el gráfico orientado hacia los algoritmos de diseño.
org.netbeans.api.visual.laf	Este paquete contiene LookFeel clase con

	estilo de definición de colores y fronteras.
org.netbeans.api.visual.layout	Este paquete contiene Layout de interfaz que define un widget-diseño.
org.netbeans.api.visual.model	Este paquete contiene ObjectScene clase que es un escenario con capacidad de registro de modelos de objetos con los widgets de la escena.
org.netbeans.api.visual.print	Este paquete contiene una sola clase ScenePrinter utiliza para imprimir una Scene de acuerdo a diversas limitaciones.
org.netbeans.api.visual.router	Este paquete contiene Router interfaz que define un router para ConnectionWidget.
org.netbeans.api.visual.vmd	Este paquete contiene un estilo de visualización VMD.
org.netbeans.api.visual.widget	Este paquete contiene Widget clase.
org.netbeans.api.visual.widget.general	Este paquete contiene generales de alto nivel widgets.

Ficheros.

Codehaus ha liberado la versión 1.3 de XStream, una librería que permite transformar objetos de Java a XML y viceversa con una gran sencillez. Permite realizar este mapeo con tan sólo tres líneas de código fuente y sin definir ningún tipo fichero de mapeo, ni emplear un compilador de esquema o DTD. Para conseguir esta serialización semimágica se basa en el API de reflection de Java.

Las mejoras respecto a la versión 1.0 son el soportar serialización personalizada llamando areadObject (), writeObject (), readResolve() y writeReplace() si están definidos, soporta ObjectInputStream.getFields() y ObjectOutputStream.putFields() en la serialización personalizada, y lee y escribe directamente a muchas APIs XML Java DOM, DOM4J, JDOM, XOM, Electric XML, StAX, Trax y, SAX. (Javahispano, 2009)

Fundamentación de la Metodología de desarrollo. Herramientas.

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos software.
Las técnicas indican cómo debe ser realizada una actividad técnica determinada identificada en la metodología. Combina el empleo de unos modelos o representaciones gráficas junto con el empleo de procedimientos detallados. Se debe tener en consideración que una técnica determinada puede ser utilizada en una o más actividades de la metodología de desarrollo de software. Además se debe tener mucho cuidado cuando se quiere cambiar una técnica por otra o de seleccionar una técnica determinada.

Durante los últimos años se han desarrollado dos corrientes en lo referente a las metodologías de desarrollo de software, las llamadas “pesadas” y las llamadas “ligeras o ágiles”. Las primeras se basan en la idea de conseguir el objetivo común por medio de orden y documentación, mientras que las segundas tratan de lograrlo por medio de la comunicación directa e inmediata entre las personas que intervienen en el proceso.

Entre las metodologías de desarrollo a analizar para seleccionar o usar para el sistema a construir tenemos:

Programación Extrema (XP) y Rational Unified Process (RUP).

Vamos a enumerar las principales diferencias respecto de las metodologías tradicionales (“no ágiles”). La siguiente tabla recoge esquemáticamente estas diferencias que no se refieren sólo al proceso en sí, sino también al contexto de equipo y organización que es más favorable a cada uno de estas filosofías de procesos de desarrollo de software. Se decide investigar sobre dos de las más conocidas de cada uno de los tipos de metodologías las agiles y las tradicionales.

Tabla 2: Diferencias entre metodologías ágiles y no ágiles

Metodología Ágil	Metodología No Ágil (Tradicional)
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo	Impuestas externamente

de desarrollo)	
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Teniendo en cuenta lo antes expuesto se va a dar un pequeño recorrido por dos de las principales metodologías.

Rational Unified Process (RUP):

El Proceso Unificado de Rational (RUP) describe como aplicar efectivamente enfoques comprobados comercialmente para el desarrollo de software. Estos enfoques son llamados "mejores prácticas" pues son utilizados en la industria por organizaciones exitosas.

RUP provee a cada miembro del equipo de las guías de proceso, plantillas y mentores de herramientas necesarios para que el equipo completo tome ventaja de, entre otras, las siguientes mejores prácticas:

Desarrollar software iterativamente con la metodología RUP

En función de la cada vez mayor complejidad solicitada para los sistemas de software, ya no es posible trabajar secuencialmente: definir primero el problema completo, luego diseñar toda la solución, construir el software y finalmente, testear el producto. Es necesario un enfoque iterativo, que permita una comprensión creciente del problema a través de refinamientos sucesivos, llegando a una solución efectiva luego de múltiples iteraciones acotadas en complejidad.

Administrar los requerimientos

Los requerimientos son las condiciones o capacidades que el sistema debe conformar. La Administración de Requerimientos es un enfoque sistemático para hallar, documentar, organizar y monitorear los requerimientos cambiantes de un sistema.

La Administración de Requerimientos permite:

- Que las comunicaciones estén basadas en requerimientos claramente definidos.
- Que los requerimientos puedan ser priorizados, filtrados y monitoreados.
- Que sea posible realizar evaluaciones objetivas de funcionalidad y performance.
- Que las inconsistencias se detecten más fácilmente.

RUP describe como:

Obtener, organizar y documentar la funcionalidad y restricciones requeridas. Documentar y monitorear las alternativas y decisiones.

Las nociones de Casos de Uso y de Escenarios utilizadas en RUP han demostrado ser una manera excelente de capturar los requerimientos funcionales y asegurarse que direccionan el diseño, la implementación y la prueba del sistema, logrando así que el sistema satisfaga las necesidades del usuario.

Utilizar arquitecturas basadas en componentes

El proceso de software debe focalizarse en el desarrollo temprano de una arquitectura robusta ejecutable, antes de comprometer recursos para el desarrollo en gran escala. RUP describe como diseñar una arquitectura flexible, que se acomode a los cambios, comprensible intuitivamente y promueve una más efectiva reutilización de software. Soporta el desarrollo de software basado en componentes: módulos no triviales que completan una función clara. RUP provee un enfoque sistemático para definir una arquitectura utilizando componentes nuevos y preexistentes.

Modelizar software visualmente

RUP muestra como modelizar software visualmente para capturar la estructura y comportamiento de arquitecturas y componentes. Las abstracciones visuales ayudan a comunicar diferentes aspectos del software; comprender los requerimientos, ver cómo los elementos del sistema se relacionan entre sí, mantener la consistencia entre diseño e implementación y promover una comunicación precisa. El estándar UML

(Lenguaje de Modelado Unificado), creado por Rational Software, es el cimiento para una modelización visual exitosa.

Verificar la calidad de software

Es necesario evaluar la calidad de un sistema respecto a sus requerimientos de funcionalidad, confiabilidad y performance. La actividad fundamental es el testing, que permite encontrar las fallas antes de la puesta en producción. RUP asiste en el planeamiento, diseño, implementación, ejecución y evaluación de todos estos tipos de testing.

El aseguramiento de la calidad se construye dentro del proceso, en todas las actividades, involucrando a todos los participantes, utilizando medidas y criterios objetivos, permitiendo así detectar e identificar los defectos en forma temprana.

Controlar los cambios al software

La capacidad de administrar los cambios es esencial en ambientes en los cuales el cambio es inevitable. RUP describe como controlar, rastrear y monitorear los cambios para permitir un desarrollo iterativo exitoso. Es también una guía para establecer espacios de trabajo seguros para cada desarrollador, suministrando el aislamiento de los cambios hechos en otros espacios de trabajo y controlando los cambios de todos los elementos de software (modelos, código, documentos, etc.). Describe como automatizar la integración y administrar la conformación de releases. (Help, 2005)

Programación Extrema (XP)

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores y propiciando un buen clima de trabajo.

XP se basa en retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

El desarrollo bajo XP tiene características que lo distinguen claramente de otras metodologías:

- Los diseñadores y programadores se comunican efectivamente con el cliente y entre ellos mismos.
- Los diseños del software se mantienen sencillos y libres de complejidad o pretensiones excesivas.
- Se obtiene retroalimentación de usuarios y clientes desde el primer día gracias a las baterías de pruebas.
- El software es liberado en entregas frecuentes tan pronto como sea posible.
- Los cambios se implementan rápidamente tal y como fueron sugeridos.
- Las metas en características, tiempos y costos son reajustadas permanentemente en función del avance real obtenido.
- Empieza en pequeño y añade funcionalidad con retroalimentación continua
- El manejo del cambio se convierte en parte sustantiva del proceso.
- El costo del cambio no depende de la fase o etapa.
- No introduce funcionalidades antes que sean necesarias.
- El cliente o el usuario se convierte en miembro del equipo.

Valores que promueve

Los valores representan aquellos aspectos que se consideran fundamentales para garantizar el éxito de un proyecto de desarrollo de software.

Los cuatro valores de XP son:

Comunicación

Existen problemas en los equipos de desarrollo por falta de comunicación, por no comentar un cambio crítico en el diseño, por no preguntar lo que se piensa al cliente. La mala comunicación no surge por casualidad y hay circunstancias que conducen a la ruptura de la misma, como aquel jefe de proyecto que reprende al programador cuando éste le comunica que hay un fallo en el diseño. XP ayuda mediante sus prácticas a fomentar la comunicación.

Coraje

El coraje es un valor muy importante dentro de la programación extrema. Un miembro de un equipo de desarrollo extremo debe tener el coraje de exponer sus dudas, miedos, experiencias sin embellecer éstas de ninguna manera. Esto es muy importante ya que un equipo de desarrollo extremo se basa en la confianza para con

sus miembros. Detrás de este valor se encuentra el lema "si funciona, mejóralo", que choca con la práctica habitual de no tocar algo que funciona, por si acaso.

Simplicidad

Dado que no se puede predecir cómo va a ser en el futuro el software que se está desarrollando; un equipo de programación extrema intenta mantener el software lo más sencillo posible. Esto quiere decir que no se va a invertir ningún esfuerzo en hacer un desarrollo que en un futuro pueda llegar a tener valor. En el XP frases como "...en un futuro vamos a necesitar..." no tienen ningún sentido ya que no aportan ningún valor en el momento. La simplicidad y la comunicación se complementan, cuanto más simple es tu sistema menos tienes que comunicar de él.

Retroalimentación

La retroalimentación actúa junto con la sencillez y la comunicación, cuanto mayor es la retroalimentación más fácil es la comunicación. Cuanto más simple un sistema, más fácil de probar. Escribir pruebas nos orienta como simplificar un sistema, hasta que las pruebas funcionen, cuando las pruebas funcionen habrá un gran avance. Esta retroalimentación se toma del cliente, de los miembros del equipo, en cuestión de todo el entorno en el que se mueve un equipo de desarrollo ágil.

Roles en XP

Programador: El programador escribe las pruebas unitarias y produce el código del sistema.

Cliente: Escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio.

Encargado de pruebas (Tester): Ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.

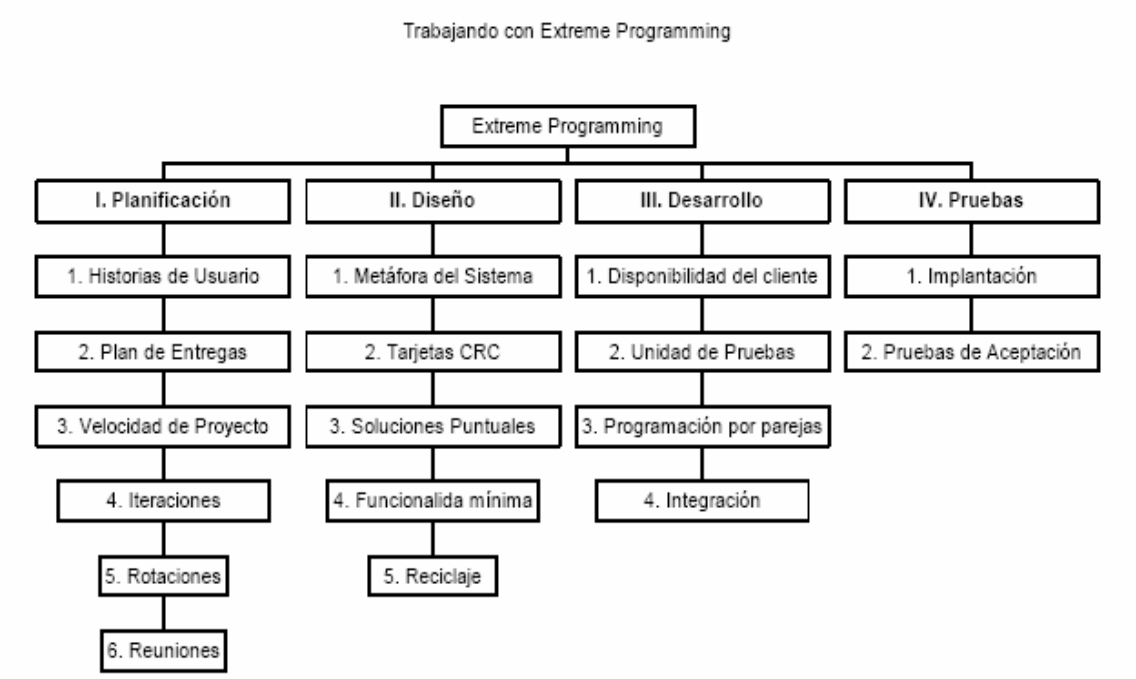
Encargado de seguimiento (Tracker): Proporciona realimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.

Entrenador (Coach): Es responsable del proceso global. Debe proveer guías al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.

Consultor: Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.

Gestor: Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es la coordinación.

Ilustración 4: **Representación de las fases de la Metodología XP**



Después de haber analizado ambas metodologías se llega a la siguiente conclusión:

- La Metodología RUP es más adaptable para proyectos de largo plazo.
- La Metodología XP en cambio, se recomienda para proyectos de corto plazo.
- Es uno de los puntos fuertes de la programación extrema. 2 personas: uno programa y el otro revisa.
- XP es una metodología que se adapta a los cambios fácilmente ya que se hace muchas iteraciones del sistema obteniendo varias versiones operativas del sistema. (Joskowicz, 2008) (Rodríguez Corbea, 2007)

Lenguaje de Modelado.

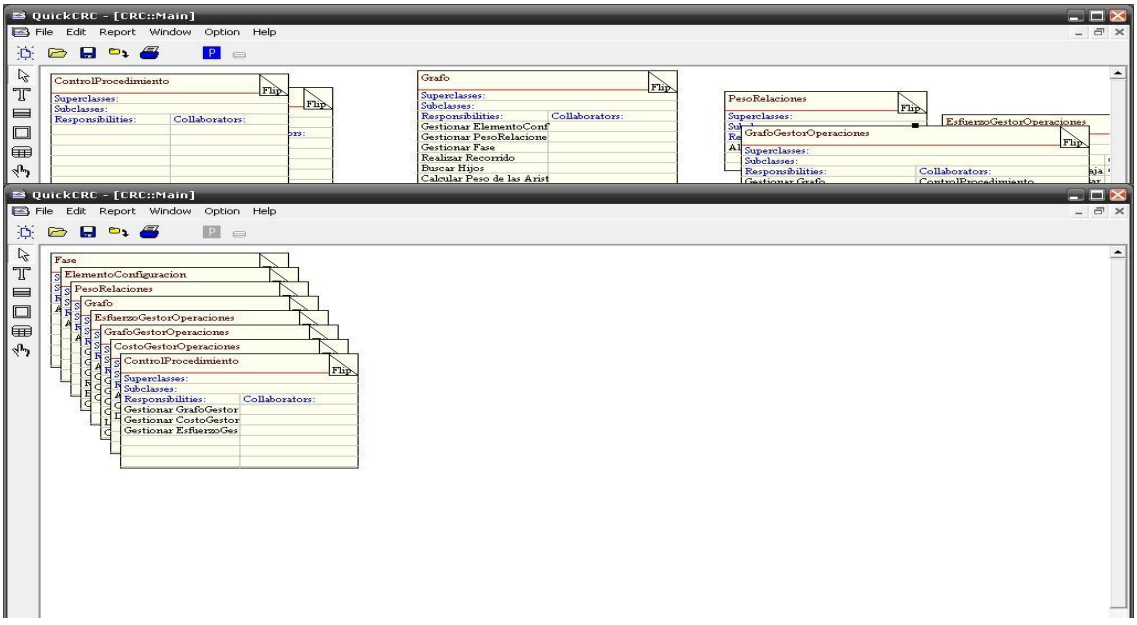
Los sistemas construidos con programación orientada a objetos (POO) proponen un mejor diseño, más comprensible, del entorno de desarrollo. Conocer un lenguaje orientado a objetos es un primer paso necesario pero insuficiente para crear sistemas de objetos. Se requiere además analizar y diseñar desde la perspectiva de los objetos. UML (Unified Modeling Languaje) es el Lenguaje Unificado de Construcción de

Modelos con que se construyen sistemas por medio de conceptos orientados a objetos. El mismo ofrece un estándar de modelado para procesos de negocio, funciones del sistema, aspectos bien definidos como esquemas de bases de datos, expresiones de lenguajes de programación y componentes de software reutilizables. Con UML se pueden modelar desde complejos sistemas de software para instituciones hasta aquellos basados en web o de tiempo real. Es un lenguaje muy expresivo, que cubre todas las vistas necesarias para desarrollar un sistema.

QuickCRC.

QuickCRC es una herramienta para el diseño con enfoque en la responsabilidad de programas "orientados a objetos" utilizando tarjetas CRC. Las tarjetas CRC son utilizadas para descubrir y documentar clases, responsabilidades, atributos y colaboraciones entre clases. Escenarios de diseño pueden ser identificados y simulados. Diseños complejos pueden ser particionados en múltiples diagramas con navegación sencilla a través de la vista de contenidos. El gráfico de herencia muestra automáticamente la estructura de clases del diseño en evolución. El usuario puede asignar el acceso de atributos de cada responsabilidad de tarjeta y la herramienta presenta un gráfico de acceso de atributos. (excelsoftware, 2009)

Ilustración 5: QuickCRC



Patrones de diseño.

Los patrones del diseño tratan los problemas del diseño que se repiten y que se presentan en situaciones particulares del diseño, con el fin de proponer soluciones a ellas. Por lo tanto, los patrones de diseño son soluciones exitosas a problemas comunes. Existen muchas formas de implementar patrones de diseño. Los detalles de las implementaciones son llamadas estrategias.

Los patrones GRASP describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable.

Se pueden destacar 5 patrones Principales que son:

- Experto.
- Creador.
- Alta cohesión.
- Bajo acoplamiento.
- Controlador.

El grupo de GoF clasificaron los patrones en 3 grandes categorías basadas en su propósito: creacionales, estructurales y de comportamiento.

Creacionales: Patrones creacionales tratan con las formas de crear instancias de objetos. El objetivo de estos patrones es de abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados.

Estructurales: Los patrones estructurales describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser incluso objetos simples u objetos compuestos.

Comportamiento: Los patrones de comportamiento nos ayudan a definir la comunicación e iteración entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre los objetos.

Estos patrones de diseño en concreto son tan importantes que ya vienen incorporados en Java. Otros vienen incluidos en otros lenguajes, tal vez algunos nunca lleguen a estar incorporados a ningún lenguaje, pero continúan siendo útiles.

Los patrones de diseño pueden incrementar o disminuir la capacidad de comprensión de un diseño o de una implementación, disminuirla al añadir accesos indirectos o aumentar la cantidad de código, disminuirla al regular la modularidad, separar mejor los conceptos y simplificar la descripción. Una vez que aprenda el vocabulario de los patrones de diseño le será más fácil y más rápido comunicarse con otros individuos que también lo conozcan.

Los patrones de diseño pueden parecerle abstractos a primera vista o, tal vez, no tenga la seguridad de que se ocupan del problema que le interesa. Comenzará a apreciarlos a medida que construya y modifique sistemas más grandes. (Larman, TOMO I)

Calidad. JUnit.

JUnit es un framework de código abierto para pruebas en java originalmente escrito por Kent Beck y Erich Gamma. Con JUnit se logra ejecutar pruebas automatizadas de manera fácil, permite evaluar si el funcionamiento de uno o varios métodos se comporta como se espera, en función de algún valor de entrada se evalúa el valor de retorno para verificar que sea el esperado. Se integra con muchos IDEs como NetBeans, BlueJ, IntelliJ, JBuilder, Eclipse y otros.
Y estos son algunos de los beneficios que ofrece a los desarrolladores:

- Documentación del código: las pruebas en sí representan documentación del código pues se aprecia cómo se debe utilizar.
- Mejor localización de errores: es más fácil detectar un error durante la realización de pruebas a un módulo que tracear todo el sistema en su búsqueda.
- Fomento del cambio: las pruebas de unidad posibilitan que el programador optimice su código o que una vez detectado un error, pueda corregirlo y probarlo nuevamente.
- Simplificación de la Integración: las *Pruebas de Integración* se realizarían mucho más fáciles y con mayor seguridad si la unidad está libre de errores. (Javahispano, 2009)

Conclusión parcial del capítulo.

En este capítulo se llevó a cabo una profundización en los elementos más importantes de la fundamentación teórica del tema, explicando los términos básicos que hacen comprensible la investigación, además de que se expusieron las normas, estándares, herramientas, técnicas y metodologías que permitirán el desarrollo de la aplicación. Debido a la investigación realizada se llegó a la conclusión de utilizar como metodología de desarrollo Programación Extrema, como lenguaje de programación Java e ID de desarrollo NetBeans por tener una gran cantidad de características importantes expuestas en los epígrafes anteriores, para el desarrollo de dicha herramienta.

Capitulo 2: Planificación y Diseño del Sistema.

Introducción.

En el presente capítulo se describirán brevemente las funciones principales y el flujo actual de los procesos involucrados en el problema en cuestión, haciéndose un análisis crítico de cómo se ejecutan actualmente estos procesos.

Además en el desarrollo de la aplicación se detallarán los pasos de la metodología que se propone en el capítulo anterior (XP). Se incluirá la elaboración de la Tarjeta de Historias de Usuarios, los requerimientos funcionales y no funcionales de la aplicación, Historia de Usuario Dividido en Tareas, Tareas de Ingeniería, Tarjetas de CRC (Clase, Responsabilidad y Colaboración), Patrones de Diseño así como la arquitectura usada.

Descripción del Sistema.

TODCOCAP (Toma de Decisiones para la Gestión de Cambios Procedimiento Automatizado) es un sistema, como lo dice su nombre en siglas que automatiza un procedimiento para la toma de decisiones en el proceso de gestión de cambios ya existente, véase (epígrafe 1.4). Para la visualización del grafo de elementos de configuración se usa la librería org.netbeans.api.visual.vmd, esta proporciona las funcionalidades de dibujar nodos, y aristas entres estos, así como de convertir esta representación en un componente, y estos a su vez en componentes dentro de este componente que apoyados por su negocio respectivo hacen de la aplicación una informatización respetable del procedimiento en cuestión, para que persistan los datos necesarios y que se permita las funcionalidades de salvar y cargar se usa la librería XStream que permite transformar objetos de Java a XML y viceversa.

Requisitos Funcionales.

- El sistema debe permitir crear Proyecto, donde el usuario debe ponerle un nombre válido.
- Una vez creado un proyecto el usuario debe poder gestionar Elementos de Configuración.
- Una vez creado un proyecto el usuario debe poder gestionar fases.
- Una vez creado un proyecto y haber insertado al menos un elemento de Configuración el usuario debe poder seleccionar y deseleccionar el elemento de configuración a cambiar especificando el porciento que cambia.
- El usuario una vez hecho un cambio en algún elemento de configuración debe poder ver un reporte general de lo que repercute este cambio en el todo en general.

- El usuario una vez hecho un cambio en algún elemento de configuración debe poder ver un análisis de esfuerzo que trae consigo ese cambio para el todo insertando la cantidad de trabajadores que estarán involucrados en el cambio.
- El usuario una vez hecho un cambio en algún elemento de configuración debe poder ver un análisis de costo que trae consigo el cambio para el todo insertando los principales costos así como, el presupuesto del proyecto y la fecha que se comenzará a trabajar en cambio así como la del fin de proyecto, además de ver si el proyecto puede realizar el cambio de acuerdo a costos materiales y de tiempo.

Requisitos No Funcionales.

- El sistema al ser desarrollado en java requiere la máquina virtual de java 1.6 o superior para su ejecución.
- Requiere de al menos 512 RAM para su correcto rendimiento y de 50MB libre en disco duro para su almacenamiento.
- El sistema debe tener una interfaz amigable.
- El sistema debe Salvar y Cargar Proyectos.
- El sistema debe tener accesos de teclado para las diferentes funcionalidades.
- El sistema debe ser multiplataforma.

Iteraciones.

Iteración 1.

Esta es la primera iteración del proyecto donde se llevará a cabo lo relacionado con gestionar los elementos de valor del sistema, dígase con esto los almacenables y sobre los cuales se va a trabajar.

Las tareas Guardar Proyecto y Abrir Proyecto se pospusieron para la tercera iteración debido a que se debería conocer todas las características de los elementos a almacenar y para ello se requiere terminar las dos primeras iteraciones.

Historias De Usuario.

Tabla 3: Gestionar Proyecto

	Historia de Usuario
Número: 1	**Usuario: Dayanis Quintana Torres**
Nombre historia: Gestionar Proyecto	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Baja

Puntos estimados: 4	**Iteración asignada: 1**
Programador responsable: Joel Jimenez - Rayner Toledo	
Descripción: Se selecciona "**Crear Proyecto**" y se introduce un nombre válido para el proyecto. Para cerrarlo se da clic en el botón. Cerrar todo o en el menú con el mismo nombre y la aplicación pregunta si se está seguro. Para salvar se da clic el botón **guardar proyecto** o en el menú con el mismo nombre, el sistema te pregunta dónde y lo salva con el mismo nombre del proyecto. Para cargar si hay algún proyecto abierto el sistema te pregunta si desea cerrarlo si aceptas lo cierra y el sistema te pregunta qué proyecto desea cargar. En caso de haber problemas en la carga o en la apertura del proyecto el sistema lanzara un mensaje de error.	
Observaciones: Las tareas Guardar Proyecto y Abrir Proyecto se pospusieron para la tercera iteración debido a que se debería conocer todas las características de los elementos a almacenar y para ello se requiere terminar las dos primeras iteraciones.	

Tabla 4: Gestionar Elemento de Configuración

		Historia de Usuario
Número: 2	**Usuario: Dayanis Quintana Torres**	
Nombre historia: Gestionar Elemento de Configuración.		
Prioridad en negocio: Alta		**Riesgo en desarrollo:** Baja
Puntos estimados: 4		**Iteración asignada: 1**
Programador responsable: Joel Jimenez - Rayner Toledo		
Descripción: Una vez creado un proyecto se selecciona un EC de la paleta de componentes y se arrastra en el área de trabajo donde deberá aparecer este con sus valores iníciales por defecto. Una vez allí se le podrá dar doble clic para editar o simplemente clic derecho que deberá salir un menú con una de las opciones que sea editar, donde deberá salir una pantalla con los valores a editar. Para eliminar se debe seleccionar la opción eliminar y luego marcar al EC a eliminar, clic derecho donde deberá salir un menú con la opción de eliminar o simplemente dar shift clic izquierdo. Una vez eliminado se deberán eliminar automáticamente las relaciones que tienen que ver con él.		
Observaciones:		

Tabla 5: Gestionar Relaciones

	Historia de Usuario
Número: 3	**Usuario: Dayanis Quintana Torres**
Nombre historia: Gestionar Relaciones	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Baja
Puntos estimados: 4	**Iteración asignada: 1**
Programador responsable: Joel Jiménez - Rayner Toledo	
Descripción: Una vez creado un proyecto y EC se selecciona el tipo de relación que se desea insertar después el inicio y el fin, si ya existe una relación entre ese inicio y fin el sistema deberá preguntar si desea remplazarla. Para eliminar se podrá seleccionar la opción eliminar o simplemente shift clic encima de la relación.	
Observaciones:	

Tabla 6: Gestionar Fases

	Historia de Usuario
Número: 4	**Usuario: Dayanis Quintana Torres**
Nombre historia: Gestionar Fases	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Baja
Puntos estimados: 4	**Iteración asignada: 1**
Programador responsable: Joel Jiménez - Rayner Toledo	
Descripción: Una vez creado un proyecto se podrá editar fases, un botón en la barra de herramientas que lleve a una pantalla con las fases existentes donde se podrá editar o eliminar. Para adicionar se deberá hacer desde editar EC cuando se selecciona a la fase que pertenece si no existe, se adiciona.	
Observaciones:	

Iteración 2.

Esta es la segunda iteración del proyecto donde se manejará lo referente con la acción cambio de ahí su importancia y peso para el proyecto.

Historia de Usuario.

Tabla 7: Gestionar Cambio

Historia de Usuario	
Número: 5	**Usuario: Dayanis Quintana Torres**
Nombre historia: Gestionar Cambio	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 4	**Iteración asignada: 2**
Programador responsable: Joel Jimenez - Rayner Toledo	
Descripción: Una vez creado un proyecto y EC se podrá realizar un cambio en este seleccionando la acción cambio y dando clic sobre este o simplemente dando clic derecho donde deberá salir un menú con esta opción, luego aparecerá una pantalla para especificar el porciento que cambia, una vez cambiado se deberá expandir el cambio hacia los demás EC relacionados, solo se podrá hacer un cambio por proyecto, para deshacer el cambio se deberá seleccionar la acción deshacer cambio y dar clic un EC cambiado o dar clic derecho en este, donde deberá salir la opción de deshacer cambio, un vez deshecho se deberá expandir la acción a los elementos de configuración relacionados y si es el elemento que originó el cambio se debe permitir realizar un nuevo cambio.	
Observaciones:	

Iteración 3.

Esta es la tercera iteración del proyecto donde se manejarán los reportes y datos de salida que dicta el procedimiento en cuestión.

Historias de Usuario.

Tabla 8: Mostrar Reporte General

	Historia de Usuario
Número: 6	**Usuario: Dayanis Quintana Torres**
Nombre historia: Mostrar Reporte General	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Baja
Puntos estimados: 4	**Iteración asignada: 3**
Programador responsable: Joel Jimenez - Rayner Toledo	
Descripción: Una vez creado un proyecto y un EC cambiado se podrá pedir al sistema que muestre un reporte general del cambio. Se mostrará en una consola.	
Observaciones:	

Tabla 9: Mostrar Evaluación de Esfuerzo

	Historia de Usuario
Número: 7	**Usuario: Dayanis Quintana Torres**
Nombre historia: Mostrar Evaluación de Esfuerzo	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Baja
Puntos estimados: 4	**Iteración asignada: 3**
Programador responsable: Joel Jimenez - Rayner Toledo	
Descripción: Una vez creado un proyecto y un EC cambiado se podrá pedir al sistema que muestre un reporte general del Esfuerzo introduciendo en una pantalla la cantidad de trabajadores involucrados en el cambio. Se mostrará en una consola.	
Observaciones:	

Tabla 10: Mostrar Evaluación de Costo

	Historia de Usuario
Número: 8	**Usuario: Dayanis Quintana Torres**
Nombre historia: Mostrar Evaluación de Costo	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Baja
Puntos estimados: 4	**Iteración asignada: 3**
Programador responsable: Joel Jimenez - Rayner Toledo	
Descripción: Una vez creado un proyecto y un EC cambiado se podrá pedir al sistema que muestre un reporte general del Costo introduciendo en una pantalla los costos materiales así como las fechas de inicio del cambio y fin del proyecto. Se mostrará en una consola.	
Observaciones:	

Plan de Entrega.

Se definirán los conjuntos de historias de usuario a implementar en cada iteración, y sus fechas de entrega.

Iteración 1.

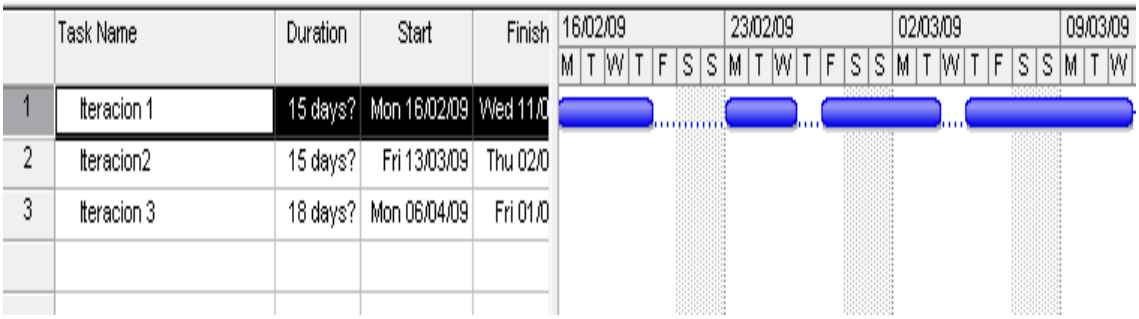
Gestionar Proyecto.

Gestionar Elemento de Configuración.

Gestionar Relaciones.

Gestionar Fases.

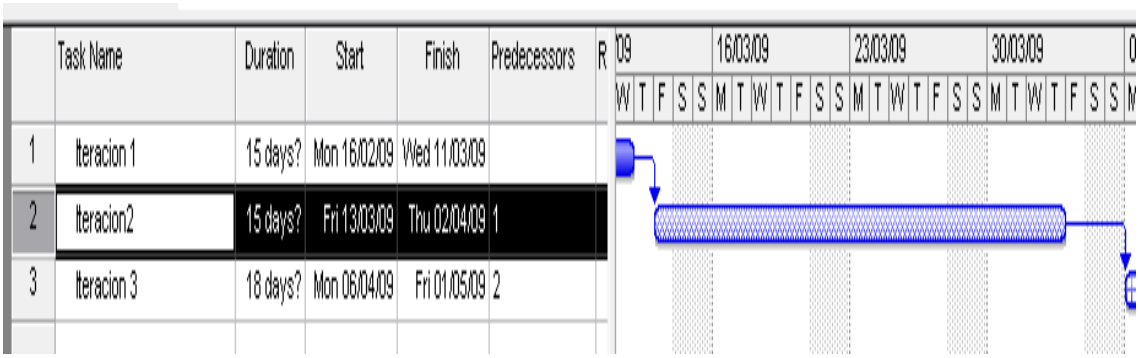
Ilustración 6: Plan de Entrega Iteración 1



Iteración 2.

Gestionar Cambio

Ilustración 7: Plan de Entrega Iteración 2



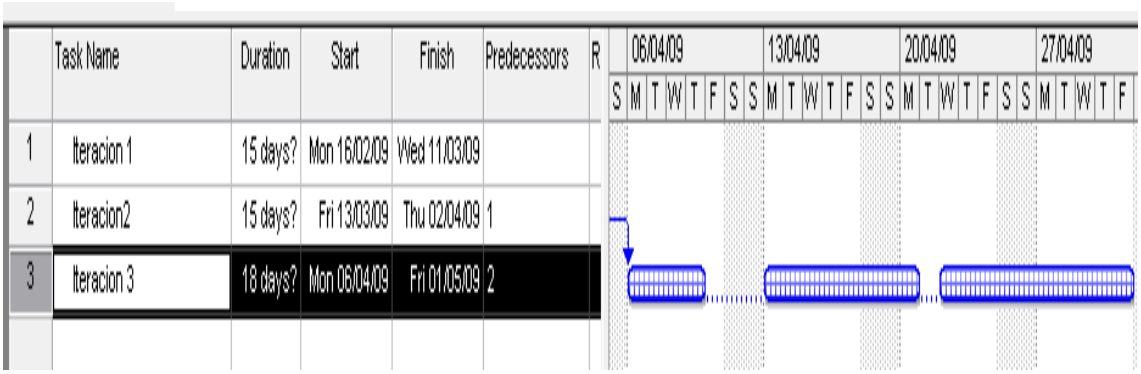
Iteración 3.

Mostrar Reporte General.

Mostrar Evaluación de Esfuerzo.

Mostrar Evaluación de Costo

Ilustración 8: Plan de Entrega Iteración 3



Historia de Usuario Dividido en Tareas.

Cada una de estas historias de usuario se transformará en tareas que serán desarrolladas por programadores, dentro del equipo de desarrollo, aplicando la práctica de la Programación en parejas. Cada tarea de desarrollo corresponderá a un período de uno a 2 días máximo de desarrollo.

Tabla 11: Historia de Usuario Dividido en Tareas

Historia de Usuario	Tareas
Gestionar Proyecto	➢ Crear Pantalla principal. ➢ Crear Área de trabajo. ➢ Crear Clases de negocio relacionadas. ➢ Crear Paleta de Componentes. ➢ Crear Consola. ➢ Crear Barra de Herramientas. ➢ Crear Barra de Menú. ➢ Administrar Componentes a visualizar. ➢ Guardar Proyecto. ➢ Abrir Proyecto.
Gestionar Elemento de Configuración (ECS).	➢ Implementar la función del componente relacionado de la paleta de componentes. ➢ Insertar ECS en el área de trabajo. ➢ Crear y habilitar opciones del menú emergente correspondiente. ➢ Crear Clases de negocio y funciones correspondientes y adicionar ECS. ➢ Crear Pantalla Editar ECS. ➢ Implementar Funciones relacionadas con Eliminar ECS. ➢ Tratar errores.
Gestionar Relaciones.	➢ Implementar la función del componente relacionado a la paleta de componentes. ➢ Insertar relación en el área de trabajo. ➢ Crear Clases de negocio correspondientes y adicionar relación. ➢ Implementar funcionalidades Eliminar. ➢ Tratar errores.
Gestionar Fases	➢ Implementar funciones visuales Gestionar Fases. ➢ Crear Pantalla Adicionar Fase.

	➢ Crear Pantalla Editar Fase. ➢ Implementar clases de negocio y funcionalidades correspondientes. ➢ Realizar Actualizaciones correspondientes. ➢ Tratar errores.
Gestionar Cambio	➢ Implementar funciones relacionadas con Crear Cambio en un ECS. ➢ Crear pantalla Porciento Cambiado. ➢ Implementar funcionalidades en clases de negocio correspondiente. ➢ Expandir cambio Actualizar ECS en la interfaz y negocio correspondiente. ➢ Administrar visualización de componentes relacionados con cambio. ➢ Implementar funcionalidades relacionadas con Deshacer Cambio. ➢ Tratar errores.
Mostrar Reporte General	➢ Mostrar reporte en la Consola. ➢ Implementar funcionalidades necesarias del negocio para calcular los datos a mostrar. ➢ Tratar errores
Mostrar Evaluación de Esfuerzo	➢ Crear pantalla introducir trabajadores involucrados en el cambio. ➢ Implementar funcionalidades necesarias del negocio para calcular los datos a mostrar. ➢ Mostrar reporte en la consola. ➢ Tratar Errores.
Mostrar Evaluación de Costo	➢ Crear Pantalla Introducir Datos relacionados con Costo. ➢ Implementar funcionalidades necesarias del negocio para calcular los datos a mostrar.

	➢ Mostrar reporte en consola. ➢ Tratar Errores.

Tareas detalladas.

Se presentan solo las tareas principales con el objetivo de no extender este epígrafe ni que el lector se sienta agobiado.

Tabla 12: Expandir cambio Actualizar ECS en la interfaz y negocio correspondiente

Tarea	
Número tarea: 4	**Número historia: 5**
Nombre tarea: Expandir cambio Actualizar ECS en la interfaz y negocio correspondiente.	
Tipo de tarea : Desarrollo	**Puntos estimados: 10**
Fecha inicio: 2 marzo 2009	**Fecha fin: 4 marzo 2009**
Programador responsable: Equipo XP	
Descripción: Una vez hecho el cambio en un ECS se debe hacer un recorrido por el grafo partiendo desde él para ver los ECS y Fases afectados por el cambio además de hacer todas las tareas colaterales que el procedimiento exige, como sumar las aristas del recorrido. Se debe mostrar en la interfaz un elemento distintivo que indique que un ECS determinado ha sido afectado por el cambio.	

Tabla 13: Implementar funcionalidades relacionadas con Deshacer Cambio

Tarea	
Número tarea: 5	**Número historia: 5**
Nombre tarea: Implementar funcionalidades relacionadas con Deshacer Cambio.	
Tipo de tarea : Desarrollo	**Puntos estimados: 10**
Fecha inicio: 13 abril 2009	**Fecha fin: 16 abril 2009**
Programador responsable: Equipo XP	

Descripción:
Si un ECS está afectado por el cambio y se selecciona deshacer cambio se deberá hacer un recorrido desde él para determinar cuáles ECS eran afectados por su cambio y eliminarlos del recorrido así como restar la suma de las relaciones correspondientes. En la interfaz se debe actualizar todo el proceso.

Tabla 14: Implementar funcionalidades necesarias del negocio para calcular los datos a mostrar

	Tarea
Número tarea: 2	**Número historia: 8**
Nombre tarea: Implementar funcionalidades necesarias del negocio para calcular los datos a mostrar.	
Tipo de tarea : Desarrollo	**Puntos estimados: 10**
Fecha inicio: 4 abril 2009	**Fecha fin: 6 abril 2009**
Programador responsable: Equipo XP	
Descripción: Se debe implementar las funcionalidades necesarias para calcular los costos materiales y de tiempo, estas se encuentran indicadas ya en el procedimiento. Teniendo en cuenta las dependencias entre las fases en caso de que existan a la hora de calcular el tiempo total. El sistema debe determinar si alcanza el presupuesto y el tiempo para realizar el cambio.	

Plan de Iteraciones.

Iteración 1

Ilustración 9: Plan de iteración 1

Historias de Usuario

1. **Gestionar Proyecto**
2. **Gestionar Elemento de Configuración.**
3. **Gestionar Relaciones**
4. **Gestionar Fases**

Asignación y estimación de las tareas

1. Crear Pantalla principal.
2. Crear Área de trabajo.
3. Crear Clases de negocio relacionadas.
4. Crear Paleta de Componentes.
5. Crear Consola.
6. Crear Barra de Herramientas.
7. Crear Barra de Menú.
8. Administrar Componentes a visualizar.
9. Guardar Proyecto.
10. Abrir Proyecto.
11. Implementar la función del componente relacionado de la paleta de componentes.
12. Insertar ECS en el área de trabajo.
13. Crear y habilitar opciones del menú emergente correspondiente.
14. Crear Clases de negocio y funciones correspondientes y adicionar ECS.
15. Crear Pantalla Editar ECS.
16. Implementar Funciones relacionadas con Eliminar ECS.
17. Implementar la función del componente relacionado de la paleta de componentes.
18. Insertar relación en el área de trabajo.
19. Crear Clases de negocio correspondientes y adicionar relación.
20. Implementar funcionalidades visuales Eliminar y de negocio Eliminar y sustituir relaciones.
21. Implementar funciones visuales Gestionar Fases.
22. Crear Pantalla Adicionar Fase.
23. Crear Pantalla Editar Fase.
24. Implementar clases de negocio y funcionalidades correspondientes.
25. Realizar Actualizaciones Correspondientes.
26. Tratar errores.

Programador 1

Tareas de la 1 a la 13

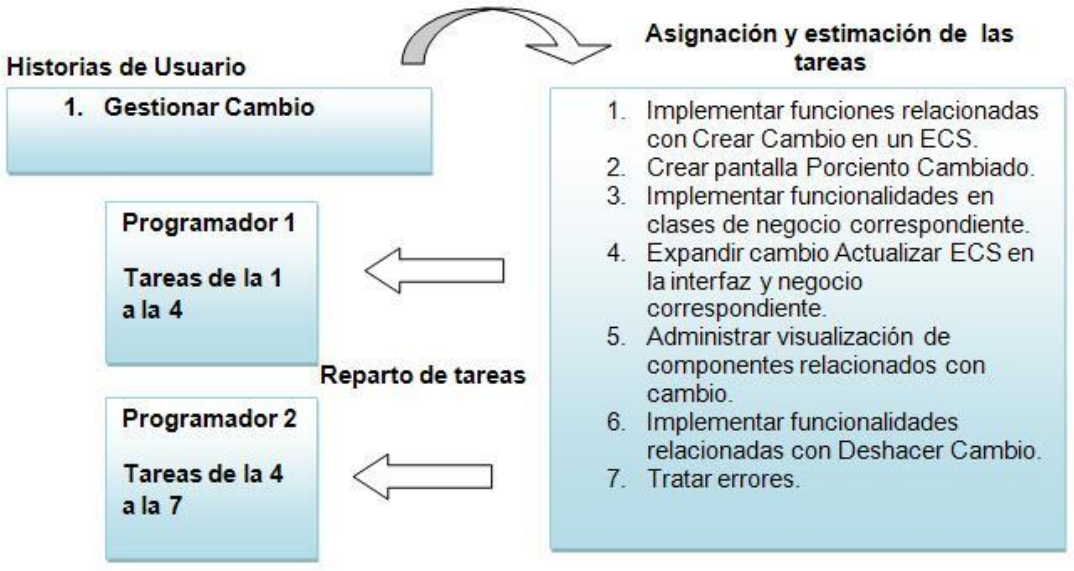
Reparto de tareas

Programador 2

Tareas de la 13 a la 26

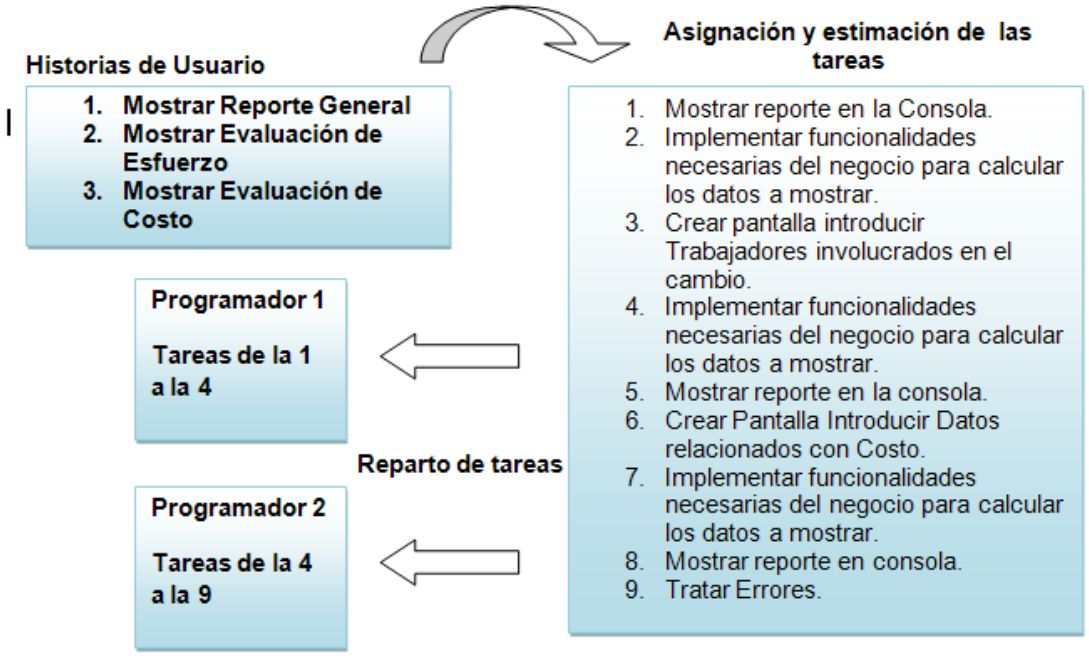
Iteración 2

Ilustración 10: Plan de iteración 2



Iteración 3

Ilustración 11: Plan de iteración 3



Tarjetas de CRC (Clase, Responsabilidad y Colaboración).

Como una extensión informal a UML, la técnica de las tarjetas CRC se puede usar para guiar el sistema a través de análisis guiados por la responsabilidad. Las clases se examinan, se filtran y se refinan en base a sus responsabilidades con respecto al sistema, y las clases con las que necesitan colaborar para completar sus responsabilidades.

Ilustración 12: Tarjeta CRC ElementoConfiguracion

ElementoConfiguracion	Flip
Superclasses:	
Subclasses:	
Responsibilities:	Collaborators:
Almacenar propiedades del ECS(nombre,peso,IdInterfaz, Fase)	Grafo

Ilustración 13: Tarjeta CRC Fase

Fase	Flip
Superclasses:	
Subclasses:	
Responsibilities:	Collaborators:
Almacenar propiedades de la Fase(id,nombre,esfuezo,dependencias)	Grafo

Ilustración 14: Tarjeta CRC PesoRelaciones

PesoRelaciones	Flip
Superclasses:	
Subclasses:	
Responsibilities:	Collaborators:
Almacenar propiedades de la Relacion(IdInterfaz,peso)	Grafo

Ilustración 15: Tarjeta CRC Grafo

Grafo	Flip
Superclasses:	
Subclasses:	
Responsibilities:	Collaborators:
Gestionar ElementoConfiguracion	
Gestionar PesoRelaciones	
Gestionar Fase	
Realizar Recorrido	
Buscar Hijos	
Calcular Peso de las Aristas del Grafo	
Buscar Nodos por Fase	GrafoGestorOperaciones

Ilustración 16: Tarjeta CRC GrafoGestorOperaciones

GrafoGestorOperaciones	Flip
Superclasses:	
Subclasses:	
Responsibilities:	Collaborators:
Gestionar Grafo	ControlProcedimiento
Gestionar Cambio	
Calcular Magnitud del Cambio	
Calcular Porciento Cambiado	
Listar Fases Involucradas en el Cambio	
Calcular Prociento Cambiado por Fase	

Ilustración 17: Tarjeta CRC EsfuerzoGestorOperaciones

EsfuerzoGestorOperaciones	Flip
Superclasses:	
Subclasses:	
Responsibilities:	Collaborators:
Almacenar cantidad de Trabajadores del Cambio	ControlProcedimiento
Calcular Esfuerzo para realizar el Cambio por Fase	
Calcular Esfuerzo para realizar el Cambio Total	
Calcular Tiempo Estimado Para realizar el Cambio	CostoGestorOperaciones

Ilustración 18: Tarjeta CRC CostoGestorOperaciones

CostoGestorOperaciones	Flip
Superclasses:	
Subclasses:	
Responsibilities:	Collaborators:
Almacenar Propiedades de Costo	ControlProcedimiento
Calcular Costo Total del Cambio	
Determinar si esta en Tiempo para Realizar el Cambio	

Ilustración 19: Tarjeta CRC ControlPreocedimientos

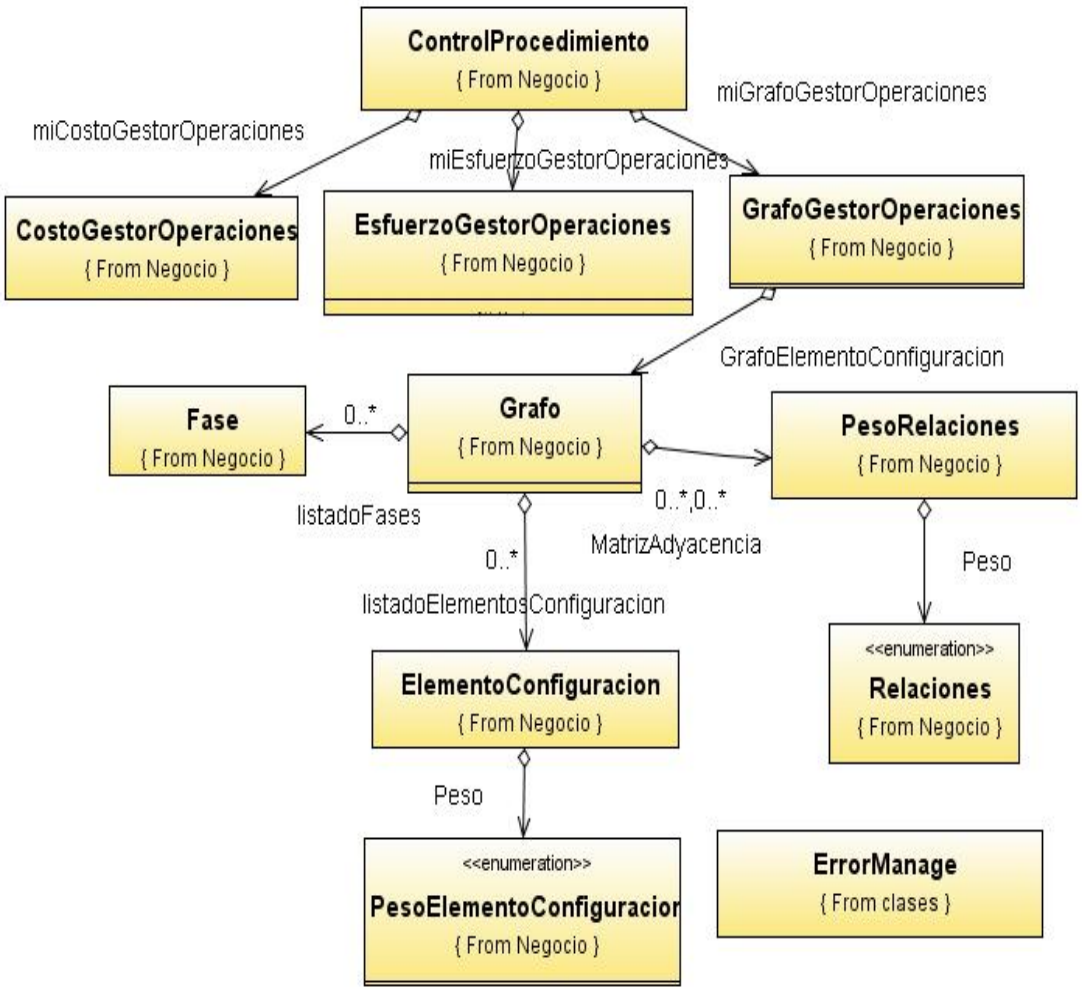
ControlProcedimiento	Flip
Superclasses:	
Subclasses:	
Responsibilities:	Collaborators:
Gestionar GrafoGestorOperaciones	
Gestionar CostoGestorOperaciones	
Gestionar EsfuerzoGestorOperaciones	

Diagrama de Clases.

Se presenta el diagrama de clases del negocio generado a partir de ingeniería inversa mostrándose la clase ControlProcedimiento, como clase controladora que supervisa y dirige dicho procedimiento haciendo uso de las clases gestoras.

No es objetivo de la metodología XP realizar el diagrama de clases pero debido a que la mayoría de los desarrolladores provienen de metodologías tradicionales y tienen la costumbre de la utilización del diagrama de clases, por lo que se mantiene el diagrama de cases para permitir un mejor tránsito hacia la nueva metodología XP.

Ilustración 20: Diagrama de Clases



Patrones de Diseño Utilizados.

Creador

La intención básica de este patrón es encontrar un Creador que necesite conectarse al objeto creado en alguna situación. Promueve el bajo acoplamiento, al hacer responsable a una clase de la creación de objetos que necesita referenciar. La creación de objetos es una de las actividades más comunes en un sistema orientado a objetos. Por lo tanto es necesario tener un principio para asignar responsabilidades de creación. El patrón Creador ofrece una respuesta a la pregunta: ¿Quién debe crear las instancias de las clases?

Si la clase A tiene agregada a la clase B, contiene instancias de B, usa muy cercanamente a B o tiene información para iniciar la clase B, entonces es responsabilidad de A de crear B.

Por ejemplo GrafoGestorOperaciones es la encargada de crear la clase Grafo pues contiene la información necesaria para hacerlo y esta a su vez crea ElementoConfiguracion, PesoReaciones y fases.

Alta Cohesión

Una clase está en alta cohesión cuando el objeto tiene delimitadas sus responsabilidades, es decir, no se le asignan responsabilidades que no estén limitadas dentro de sus funciones. Las responsabilidades de un objeto se traducen después en métodos que realizan acciones.

TODCOCAP se diseñó asignando responsabilidades de modo que la cohesión sea alta, ejemplo de esto lo representan los métodos del negocio, pues estos tienen bien delimitadas sus responsabilidades, no recargando en ningún caso sus funcionalidades, permitiendo mayor eficiencia y que el tiempo de respuesta y ejecución no exceda lo estimado, generando por ende bajo acoplamiento.

Bajo Acoplamiento.

El bajo acoplamiento hace referencia a que un objeto tenga la mínima dependencia posible con el resto del sistema. Esto posibilita que se puedan realizar modificaciones en partes del programa sin necesidad de que los objetos relacionados se vean afectados en gran medida.

Fachada.

El patrón de diseño Fachada sirve para proveer de una interfaz unificada sencilla que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más

complejas. La idea principal es la de ocultar todo lo posible la complejidad de un sistema, el conjunto de clases o componentes que lo forman, de forma que solo se ofrezca un (o unos pocos) punto de entrada al sistema tapado por la fachada. El proceso consiste en asignar la responsabilidad a una clase directa para que colabore con una clase indirecta, de modo que no se necesite saber nada de la clase indirecta. Una ventaja de usar una clase fachada para comunicar dos partes o componentes, es la de aislar los posibles cambios que se puedan producir en alguna de las partes. Si cambias el medio de comunicación o de almacenamiento de una de las partes, la otra, que hace la presentación, no tiene por qué enterarse, y viceversa.

En la solución de este sistema se utiliza este patrón Fachada para abstraer la capa de presentación de la librería de clases Org-netbeans-api-visual. Este patrón está implementado en una clase que lleva el nombre de GrafoGestorInterfas, permitiendo que la comunicación se establezca siempre con la clase GrafoGestorInterfas y no con las clases de Org-netbeans-api-visual directamente, muchas veces no conocidas en su totalidad por los desarrolladores. Además permite que el sistema sea portable a cualquier cambio en la librería de clases de Org-netbeans-api-visual, por tanto reduce el impacto al cambio y permite la escabilidad del sistema.

Estándares de Codificación.

El Estándar de Codificación definido para la realización del sistema constituye una serie de convenciones de código para el lenguaje de programación Java, con el objetivo de regular la calidad de la implementación estableciendo un estándar de desarrollo común por el cual se rige la programación del sistema informático.

El formato regula los siguientes aspectos:

- Organización de los Ficheros.
- Identación.
- Comentarios.
- Declaraciones Espacios en blanco.
- Convenciones de Declaración.
- Prácticas de Programación.

Todo lo anterior refleja características de implementación de código Java propias de un mismo programador en conformidad con la legibilidad y uniformidad a establecerse como buena práctica de programación en todo sistema automatizado.

Arquitectura en tres capas.

La capa de presentación o interfaz de usuario.

En este caso, está formada por los formularios y los controles que se encuentran en los formularios. Capa con la que interactúa el usuario.

La capa de negocio.

Esta capa está formada por las entidades empresariales, que representan objetos que van a ser manejados o consumidos por toda la aplicación. En este caso, están representados por las clases que se crean relacionadas con el procedimiento.

La capa de acceso a datos.

Contiene clases que interactúan con la fuente de almacenamiento de datos, estas clases altamente especializadas se encuentran en la arquitectura y permiten, utilizando los procedimientos almacenados generados, realizar todas las operaciones con la fuente de almacenamiento de datos de forma transparente para la capa de negocio. En el caso de la clase SerializadorManage y la librería XStream son las encargadas de serializar, almacenar y cargar toda la información necesaria.

Conclusiones del Capitulo.

En el presente capítulo se describió brevemente las funciones principales y el flujo actual de los procesos involucrados en el problema en cuestión, haciéndose un análisis crítico de cómo se ejecutan actualmente estos procesos.

Además en el desarrollo de la aplicación se detallan los pasos de la metodología que se propone en el capítulo anterior (XP), lo que permite la implementación del sistema.

Capitulo 3: Implementación y Pruebas del Sistema.

Introducción.

Como Plantea la metodología XP, el cliente es una parte más del equipo de desarrollo; su presencia es indispensable en las distintas fases de XP. A la hora de codificar una historia de usuario su presencia es aún más necesaria. No olvidemos que los clientes son los que crean las historias de usuario y negocian los tiempos en los que serán implementadas. Antes del desarrollo de cada historia de usuario el cliente debe especificar detalladamente lo que ésta hará y también tendrá que estar presente cuando se realicen los test que verifiquen que la historia implementada cumple la funcionalidad especificada.
La codificación debe hacerse ateniendo a estándares y patrones de codificación ya creados. Programar bajo estándares mantiene el código consistente y facilita su comprensión y la escalabilidad.

Crear test que prueben el funcionamiento de los distintos códigos implementados nos ayudará a desarrollar dicho código. Crear estos test antes ayuda a saber qué es exactamente lo que tiene que hacer el código a implementar y se sabrá que una vez implementado pasará dichos test sin problemas ya que dicho código ha sido diseñado para ese fin. Se puede dividir la funcionalidad que debe cumplir una tarea a programar en pequeñas unidades, de esta forma se crean primero los test para cada unidad y a continuación se desarrolla dicha unidad, así poco a poco se consigue un desarrollo que cumpla todos los requisitos especificados.

Como ya se mencionó, XP opta por la programación en pareja ya que permite un código más eficiente y con una gran calidad.

XP sugiere un modelo de trabajo usando repositorios de código dónde las parejas de programadores publican cada pocas horas sus códigos implementados y corregidos junto a los test que deben pasar. De esta forma el resto de programadores que necesiten códigos ajenos trabajarán siempre con las últimas versiones. Para mantener un código consistente, publicar un código en un repositorio es una acción exclusiva para cada pareja de programadores.

XP también propone un modelo de desarrollo colectivo en el que todos los programadores están implicados en todas las tareas; cualquiera puede modificar o ampliar una clase o método de otro programador si es necesario y subirla al repositorio

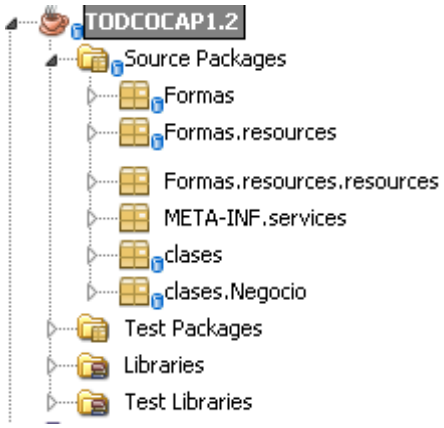
de código. Así permite al resto de los programadores modificar códigos que no son suyos no supone ningún riesgo ya que para que un código pueda ser publicado en el repositorio tiene que pasar los test de funcionamiento definidos para el mismo.

La optimización del código siempre se debe dejar para el final. Hay que hacer que funcione y que sea correcto, más tarde se puede optimizar.

Sistema de Archivos.

El sistema de archivos es un componente importante de la aplicación. Pues mantiene la organización y limpieza del proyecto.

Ilustración 21: Sistema de archivos



Se muestra la siguiente organización de paquetes. El paquete Formas almacena todas las pantallas del sistema y Formas.resourses todos los recursos que estas necesitan para su correcto funcionamiento y visualización. El paquete clases almacena las clases que se encuentran entre la interfaz y el negocio así como el paquete clases.negocio se almacenan todas las clases del negocio. Todo lo anterior se encuentra dentro de la carpeta Source Packages.

Estándares de Implementación.

Se explicará brevemente como se usaron los estándares de codificación en el desarrollo del sistema.

Tabla 15: Estándares de Implementación

Elemento	¿Cómo?	Ejemplo
Clases	➢ Se nombran con mayúscula con un nombre semejante a la función que realizan o lo que representan. ➢ Si es una palabra compuesta se usa la mayúscula para separar una de otra.	CostoGestorOperaciones ElementoConfiguracion
Identificadores	➢ Comienzan con minúscula y hacen una referencia a lo que significan. ➢ Si es una palabra compuesta se usa la mayúscula para separar una de otra.	miGrafoGestorOperaciones formaTituloP Cprocedimiento
Métodos	➢ Se nombran con mayúscula con un nombre semejante a la función que realizan o lo que representan excepto los set o get. ➢ Si es una palabra compuesta se usa la mayúscula para separarlas.	BusquedaIdInterfazNodoE(String pId) Recorrido(String IdE) PesoAristasGrafo()

Pruebas de Unidad por Clases. JUnit.

Cuando se prueba un programa, se ejecuta con unos datos de entrada (casos de prueba) para verificar que el funcionamiento cumple los requisitos esperados. Se define prueba unitaria como la prueba de uno de los módulos que componen un programa.
En los últimos años se han desarrollado un conjunto de herramientas que facilitan la elaboración de pruebas unitarias en diferentes lenguajes. Dicho conjunto se denomina XUnit. De entre dicho conjunto, JUnit es la herramienta utilizada para realizar pruebas unitarias en Java.

El concepto fundamental en estas herramientas es el caso de prueba (test case), y la suite de prueba (test suite). Los casos de prueba son clases o módulos que disponen de métodos para probar los métodos de una clase o módulo concreta/o. Así, para cada clase que se quisiera probar se definiría su correspondiente clase de caso de prueba. Mediante las suites se puede organizar los casos de prueba, de forma que cada suite agrupa los casos de prueba de módulos que están funcionalmente relacionados.

Las pruebas que se van construyendo se estructuran así en forma de árbol, de modo que las hojas son los casos de prueba, y podemos ejecutar cualquier subárbol (suite).

De esta forma, se construyen programas que sirven para probar nuestros módulos, y que se pueda ejecutar de forma automática. A medida que la aplicación vaya avanzando, se dispondrá de un conjunto importante de casos de prueba, que servirá para hacer pruebas de regresión. Eso es importante, puesto que cuando cambiamos un módulo que ya ha sido probado, el cambio puede haber afectado a otros módulos, y sería necesario volver a ejecutar las pruebas para verificar que todo sigue funcionando.

Para la construcción y desarrollo guiado por pruebas se empezará por la elaboración de pruebas de las principales funcionalidades de las clases gestoras que poseen el gran mazo de operaciones.

GrafoGestorOperaciones.
Esta clase como su nombre lo indica es la encargada de gestionar todo a lo que a grafo se refiera. Entre sus principales funcionalidades se encuentran.
Recorrido- realiza un recorrido a partir del nodo afectado por cambio por todos sus hijos e hijos de hijos.

ProcientoCambio-Calcula el porciento del cambio para el proyecto.

MagnitudCambio-Calcula la magnitud del cambio para el proyecto.

PesoAristaGrafo-Calcula el peso de las aristas del grafo.

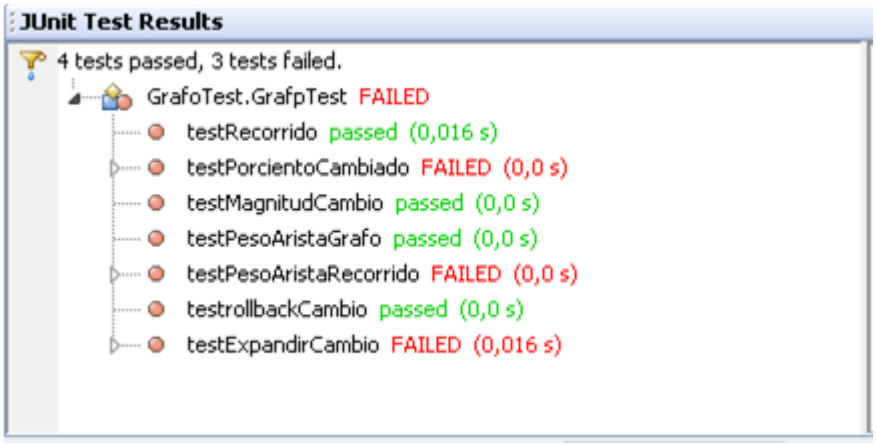
RollBackCambio-Deshace el cambio a partir de un nodo determinado.

ExpandirCambio-Expande el cambio a partir de un nodo determinado.

ListarFasesInvolucardas-Lista las Fases involucradas en el cambio.

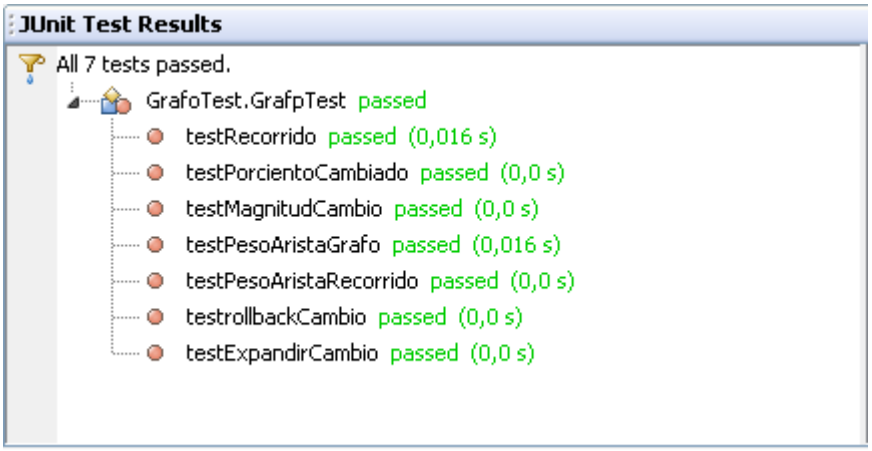
PesoRelacionesRecorrido-Valor del peso de las relaciones del recorrido.

Ilustración 22: Primera Iteración GrafoGestorOperaciones



En las primeras pruebas diseñadas y después de implementar las funcionalidades se obtuvo los resultados expuestos en la figura. Se perfeccionaron y solucionaron lo errores cometidos hasta que todas la funcionalidades de la clase lograron pasar las pruebas como se muestra en la siguiente figura.

Ilustración 23: Posteriores Iteraciones GrafoGestorOperaciones



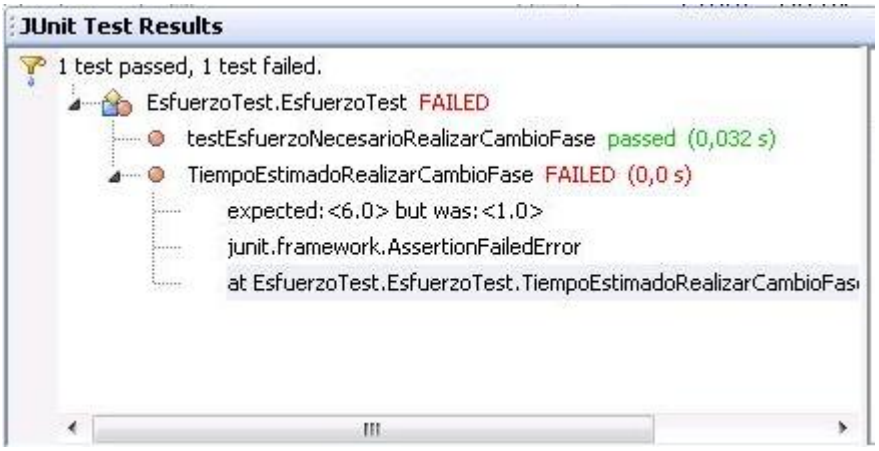
EsfuerzoGestorOperaciones.

La siguiente clase es la encargada de calcular y gestionar todas las operaciones que se relacionen con el esfuerzo requerido para realizar un cambio determinado. Entre sus principales funcionalidades se encuentran.

EsfuerzoNecesarioRealizarCambioFase-Devuelve el esfuerzo necesario para realizar el cambio por fase.

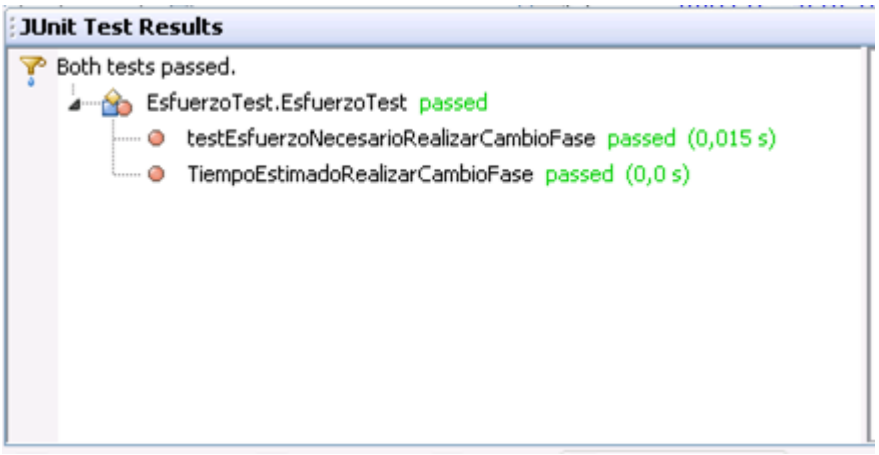
TiempoEstimadoRealizarCambioFase-Devuelve el tiempo en horas requerido para realizar el cambio por fase.

Ilustración 24: Primera Iteración EsfuerzoGestorOperaciones



Como se muestra en la figura en las primeras iteraciones se detectaron errores en TiempoEstimadoRealizarCambioFase pues la función elaborada no sobrepasó las pruebas más adelante se revisaron y solucionaron los errores cometidos hasta el obtener el resultado que se muestra a continuación.

Ilustración 25: Posteriores Iteraciones EsfuerzoGestorOperaciones



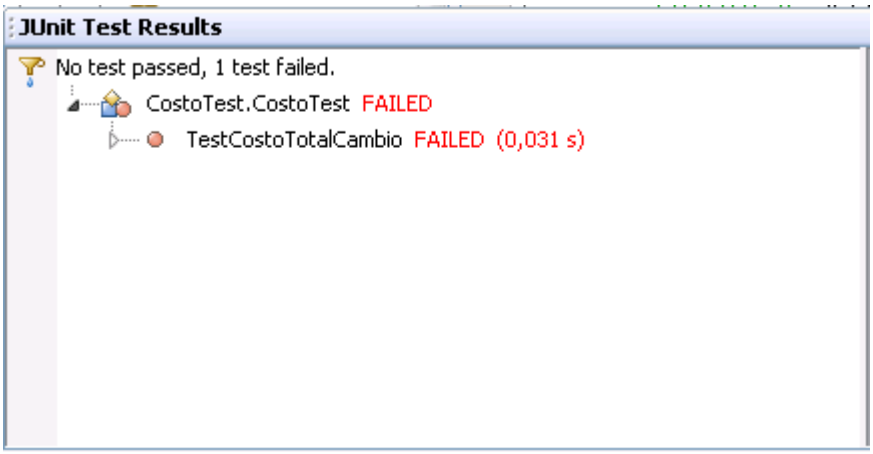
CostoGestorOperaciones.

Esta clase como anuncia es la encargada de manipular las informaciones de costo así como se responsabiliza de determinar las siguientes funcionalidades:

CostoTotalCambio- Calcula el costo material total que requiere el cambio para lograr realizarse de forma adecuada.

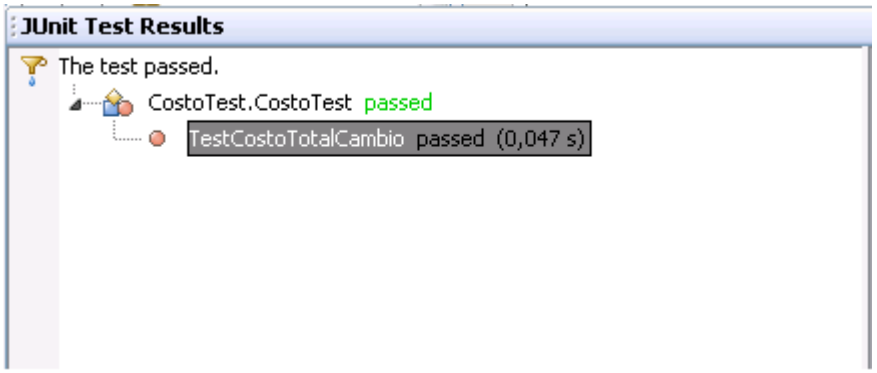
IsOnTiempo- Determina si el cambio puede realizarse en costos de tiempo (si alcanza o no el tiempo para realizar el cambio).

Ilustración 26: Primera Iteración CostoGestorOperaciones



Resultados primeras iteraciones.

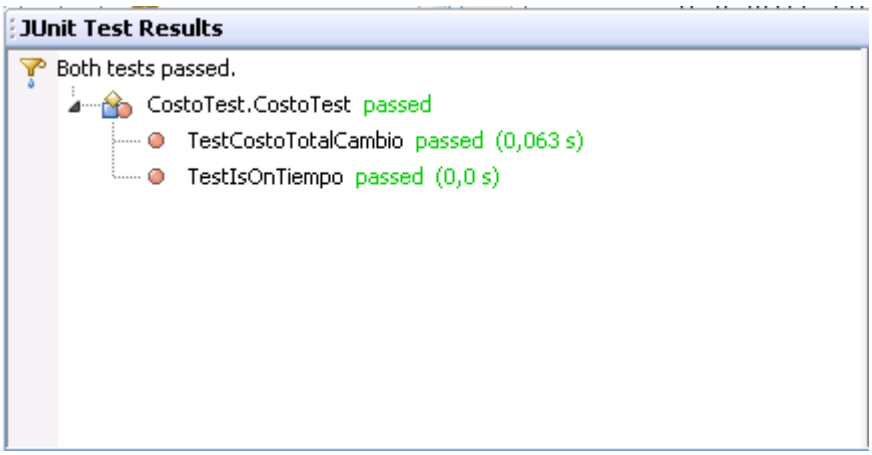
Ilustración 27: Segunda Iteración CostoGestorOperaciones



Resultado final después de corregir algunos errores.

Ahora para el método isOnTiempo la prueba a que se debía someter resultó durante gran cantidad de intentos fallida, pero con la experiencia se logró como resultando final completar su construcción.

Ilustración 28: Posteriores Iteraciones CostoGestorOperaciones



Métricas de diseño.

Con el objetivo de determinar el grado de calidad y fiabilidad del diseño que se propone se establecieron algunas métricas de diseño basadas en clases para medir categorías tales como tamaño y número de descendientes, para aspectos orientados al código, a la cohesión, al acoplamiento y la reutilización.

Métrica Tamaño de clase (TC).

El tamaño general de una clase se puede determinar empleando medidas para saber el **número total de operaciones** (tanto operaciones heredadas como privadas de la instancia) que están encapsuladas dentro de la clase así como encontrando el **número de atributos** (tanto atributos heredados como atributos privados de la Instancia) que están encapsulados en la clase. Si existen valores grandes de TC éstos mostrarán que una clase puede tener demasiada responsabilidad, lo cual reducirá la reutilizabilidad de la clase y complicará la implementación y la comprobación, por otra parte cuanto menor sea el valor medio para el tamaño, más probable es que las clases existentes dentro del sistema se puedan reutilizar ampliamente.

La mayor cantidad de clases que involucran procesos presentes en el sistema informático TODCOCAP se encuentra en la Capa de Negocios, fue a esta capa a la que se le aplicó la métrica del tamaño de clase (TC).

Tabla 16: Clases de la Capa de Negocio

Clase	Nro. De Atributos	Nro. De Operaciones
ControlProcedimiento	**3**	**21**
CostoGestorOperaciones	**7**	**2**
ElementoConfiguracion	**5**	**0**
EsfuerzoGestorOperaciones	**1**	**3**
Fase	**4**	**0**
Grafo	**6**	**18**
GrafoGestorOperaciones	**5**	**20**
PesoElementoConfiguracion	**1**	**0**
PesoRelaciones	**2**	**0**
Relaciones	**1**	**0**
Total	**32**	**43**

Teniendo en cuenta las medidas o umbrales de referencia en lo que respecta al **número de operaciones** y/o **atributos** de las clases se establece que un tamaño de clase **pequeño** es aquel que tiene un valor menor o igual que 20. Un tamaño de clase **medio** es aquel cuyos valores exceden a 20 y son menores o incluyen a 30 y un tamaño de clase **grande** es aquel que es mayor que este último valor (30). Así se concluye que la Capa de Negocio cuenta con 10 clases, para un promedio de cantidad de atributos de 3.2 y un promedio de cantidad de operaciones de 4.3.

Los valores de tamaño quedan distribuidos de la siguiente manera:

Tabla 17: Cantidad de clases por tamaño

Umbral	Tamaño	Cantidad de Clases
<= 20	**Pequeño**	**7**
>20 y <= 30	**Medio**	**3**

>30	**Grande**	**0**

La tabla de valores muestra que la mayoría de las clases son pequeñas por lo que se cumplen los parámetros establecidos por la métrica de tamaño de clase.

Número de operaciones redefinidas para una subclase (NOR).

Resultado de la aplicación: Como los umbrales definidos para esta métrica indican que valores grandes de NOR implican problemas en el diseño y violación de la abstracción de la superclase; luego de aplicada la métrica al diseño de clases, se llega a la conclusión de que los resultados obtenidos están dentro de los umbrales definidos puesto que de un total de 10 clases a las que se le aplicó la métrica, ninguna reemplaza operaciones por lo que no se remplaza ninguna operación, como se muestra en la tabla a continuación.

Tabla 18: Total de clases que redefinen operaciones

Cantidad de clases del sistema	Total de subclases que reemplazan operaciones
10	**0**

Como se muestra en la tabla, el por ciento de clases que reemplazan es mínimo en relación a las que no reemplazan, por tanto se puede concluir que la jerarquía de clases existente no es una jerarquía débil, permitiendo esto que el software pueda ser fácilmente probado y modificado, lo que implica que el tiempo requerido para los cambios no se vea afectado y que la abstracción representada por la superclase se respetó durante el diseño del sistema.

Pruebas de Interfaz.

Se refiere a las pruebas que se llevan a cabo sobre la interfaz del software, con el objetivo de demostrar que las funciones del sistema son operativas. Estas pruebas se centran fundamentalmente en los requisitos funcionales. Se realiza una muestra por historias de usuario del funcionamiento del sistema, para tener medida de la calidad y eficiencia con que cuenta.

Tabla 19: Caso de prueba: Gestionar Proyecto

Nombre de la Historia de Usuario: Gestionar Proyecto		
Nombre del Caso de prueba: Adicionar Proyecto		
Entrada	**Resultados**	**Condiciones**
➢ Se adiciona al sistema un nombre de proyecto correcto. ➢ Se deja en Blanco el nombre del proyecto.	➢ La aplicación adiciona el proyecto al área de trabajo ➢ La aplicación muestra un mensaje especificando que el nombre del proyecto no es correcto.	
Nombre del Caso de prueba: Cerrar Proyecto		
➢ El usuario selecciona la opción cerrar proyecto.	➢ El sistema muestra una forma preguntando que si no desea guardar antes de cerrar	➢ Que exista un proyecto abierto.

Tabla 20: Caso de prueba: Gestionar Fase

Nombre de la Historia de Usuario: Gestionar Fase		
Nombre del Caso de prueba: Adicionar Fase		
Entrada	**Resultados**	**Condiciones**
➢ Se adiciona una Fase insertando los datos correctamente ➢ Se adiciona la fase introduciendo datos	➢ La aplicación adiciona la fase correctamente ➢ La aplicación muestra un mensaje	

incorrectos en cada uno de sus editables. ➢ Se dejan en blanco.	especificando que no son correctos los datos insertados. ➢ La aplicación muestra un mensaje especificando que debe insertar datos.	
Nombre del Caso de prueba: Editar Fase		
➢ El usuario introduce los datos correctamente. ➢ El usuario inserta los datos incorrectamente. ➢ El usuario elimina sin existir una fase. ➢ El usuario elimina una fase existente.	➢ La aplicación edita la fase correctamente ➢ La aplicación muestra un mensaje. especificando que no son correctos los datos insertados. ➢ La aplicación muestra un mensaje especificando que no existen fases. ➢ La aplicación elimina la fase correctamente.	

Tabla 21: Caso de prueba: Gestionar Cambio

Nombre de la Historia de Usuario: Gestionar Cambio		
Nombre del Caso de prueba: Realizar Cambio		
Entrada	**Resultados**	**Condiciones**
➢ Se adiciona un Porciento de cambio introduciendo correctamente los	➢ La aplicación realiza el cambio correctamente.	

cambios. ➢ Se adiciona un Porciento de cambio introduciendo incorrectamente los cambios. ➢ Se deja en blanco el editable.	➢ La aplicación muestra un mensaje especificando que no son correctos los datos insertados. ➢ La aplicación muestra un mensaje especificando que debe insertar datos.	
Nombre del Caso de prueba: Deshacer Cambio		
➢ El usuario selecciona el elemento de configuración.	➢ La aplicación deshace el cambio correctamente.	

Tabla 22: Caso de prueba: Gestionar Esfuerzo

Nombre de la Historia de Usuario: Gestionar Esfuerzo		
Nombre del Caso de prueba: Calcular esfuerzo		
Entrada	**Resultados**	**Condiciones**
➢ Se Selecciona calcular esfuerzo. ➢ Se introducen los datos correctamente. ➢ Se introducen los datos incorrectamente.	➢ La aplicación abre una forma para introducir la cantidad de trabajadores. ➢ La aplicación muestra los datos generados en la consola de forma correcta. ➢ La aplicación muestra un mensaje especificando que	

➢ Se deja en blanco el editable.	debe insertar datos correctamente. ➢ La aplicación muestra un mensaje especificando que debe insertar datos.	

Tabla 23: Caso de prueba: Gestionar Costo

Nombre de la Historia de Usuario: Gestionar Costo		
Nombre del Caso de prueba: Calcular Costo		
Entrada	**Resultados**	**Condiciones**
➢ Se Selecciona calcular costo. ➢ Se introducen los datos correctamente. ➢ Se introducen los datos incorrectamente. ➢ Se deja en blanco el editable.	➢ La aplicación abre una forma para introducir la cantidad de trabajadores. ➢ La aplicación muestra los datos generados en la consola de forma correcta. ➢ La aplicación muestra un mensaje especificando que debe insertar datos correctamente. ➢ La aplicación muestra un mensaje especificando que debe insertar datos.	

Validación de la funcionalidad del Sistema.

En el siguiente epígrafe se mostrará la realización de un caso de estudio hecho de forma manual y por el sistema con el fin de demostrar las facilidades que este proporciona.

Caso de Estudio.

En un Proyecto de Desarrollo de Software donde se utiliza la metodología de desarrollo RUP se desean hacer algunos cambios y el equipo de trabajo tiene que realizar un análisis para decidir si es posible la realización de los mismos. Dicho proyecto se encuentra a mediados de la fase de construcción y el código fuente ya tiene funcionalidades integradas a la Línea Base. En el proceso de contratación con el cliente y durante la fase de inicio se confeccionó el Plan de Proyecto donde se pudo estimar el esfuerzo en horas–hombres que se necesita para la realización del trabajo. El Plan de Proyecto contiene además una estimación del costo que implica el desarrollo del producto y los beneficios que aportará el mismo al equipo de desarrollo. La estimación realizada a inicios del proyecto arrojó como resultados que el esfuerzo que se necesitaba para la realización del mismo seria de 1200 horas-hombre.

El equipo de desarrollo está constituido por 17 trabajadores los cuales están distribuidos de la siguiente forma:

- 1 Jefe de Proyecto
- 1 Analista de Negocio
- 2 Analistas del Sistema
- 3 Diseñadores
- 1 Arquitecto
- 4 Desarrolladores
- 1 Integrador del Sistema
- . 2 Responsables de las Pruebas
- 1 Responsable de Gestión de Configuración
- . 1 Planificador

Cada trabajador cobra $ 2.50 por hora de trabajo, cada computadora que permanezca encendida representa $0.20 por hora en gasto por corriente eléctrica, por cada jornada laboral que se utilice una computadora se depreciará $ 0.50 producto de que se estará acortando su vida útil. El presupuesto destinado al inicio del proyecto para subsidiar los cambios que sean necesarios fue de $ 250, en este momento el equipo de desarrollo cuenta con $200 porque han sido gastados $ 50. Existe un control del

tiempo real dedicado a cada flujo de trabajo ya terminado; además de los ECS identificados en ellos.

Fase1

Tabla 24: ECS del Flujo de Trabajo Modelamiento del Negocio

Modelamiento del Negocio (Duración: 4 Días) (1 trabajador)	
ECS	**Complejidad**
Modelo de Negocio.Doc	Media
Diagrama de Casos de Uso del Negocio. Diag	Media
Diagrama de Actividades. Diag	Media
Modelo de Objetos. Diag	Media
Reglas del Negocio. Doc	Baja
Documento Visión. Doc	Baja

Fase2

Tabla 25: ECS del Flujo de Trabajo Requerimientos

Requerimientos (Duración: 7 Días) (2 trabajadores)	
ECS	**Complejidad**
Especificación de Casos de Usos del Sistema. Doc	Media
Diagrama de Casos de Uso. Diag	Media
Descripción Textual de los CUS. Doc	Media
Especificación de Requerimientos. Doc	Media
Prototipo de Interfaz de Usuario. Doc	Alta
Descripción de la Arquitectura (Vista de CU). Doc	Alta

Fase3

Tabla 26: ECS del Flujo de Trabajo Análisis y Diseño

Análisis y Diseño (Duración: 10 Días) (3 trabajadores)	
ECS	**Complejidad**
Modelo de análisis.Doc	Media
Diagrama de Cases del Análisis. Diag	Media
Diagramas de Colaboración. Diag	Media
Modelo de Diseño. Doc	Media
Diagrama de Clases del Diseño. Diag	Alta
Diagrama de Secuencia. Diag	Alta
Documento de Arquitectura (Vista de diseño). Doc	Alta
Modelo de Despliegue. Diag	Alta
Documento de Arquitectura (Vista de despliegue). Doc	Alta
Modelo de Datos	Alta

Fase4

Tabla 27: ECS del Flujo de Trabajo Implementación

Implementación (Duración: 6 Días) (4 trabajadores)	
ECS	**Complejidad**
Modelo de Implementación. Doc	Alta
Diagramas de Componentes. Diag	Alta
Código Fuente.Exe	Alta
Base de Datos. Bd	Alta

El desarrollador propone realizar un cambio en la Interfaz de Usuario, esta interfaz ya se encuentra implementada y forma parte de la Versión 1.1 del código fuente que se encuentra integrado a la Línea Base. Este desarrollador revisó que la Interfaz propuesta cumpliera con los requisitos, a raíz de esta revisión comprobó que cambiando la interfaz, además de cumplir con los requisitos funcionales y no funcionales del proyecto, este cambio proporcionaría una mejor usabilidad del producto y el cumplimiento de los requisitos que representa sería más óptimo. El interesado llena una Solicitud de Cambio en la cual alega que este cambio además ayudará en la programación de las funcionalidades restantes. Con el objetivo de defender su solicitud plantea que haciendo un análisis preliminar de los ECS existentes este cambio no provocaría tantas afectaciones en la Línea Base. El ECS afectado directamente sería el Prototipo de Interfaz de Usuario. En este procedimiento se consideran como actividades de mayor importancia el análisis del Impacto de los Cambios y la decisión de aceptar o rechazar el cambio, para el cumplimiento de estas actividades se aplicará el procedimiento para la Toma de Decisiones en la Gestión de Cambio. Este procedimiento recibe como entrada la Solicitud de Cambio y el Informe de Estado del Cambio el cual fue creado en la primera actividad del procedimiento para la Gestión de Cambio. El analista del sistema que creó el Prototipo de Interfaz de Usuario realizó una evaluación del cambio que se desea hacer en este ECS y concluyó que el mismo cambiará en un 15 % en caso de que el cambio sea aprobado.

Primer Paso.

La construcción del grafo es la próxima actividad a realizar.

Para la construcción del grafo lo primero que se debe hacer es otorgar a cada ECS un identificador, en la siguiente tabla se le otorga este identificador.

Tabla 28: Elementos de Configuración con sus Identificadores y la complejidad

Elemento de Configuración de Software	Identificador	Complejidad
Modelo de Negocio.	A	2
Diagrama de Casos de Uso del Negocio.	B	2
Diagrama de Actividades.	C	2
Modelo de Objetos.	D	2
Reglas del Negocio.	E	1

Documento Visión.	F	1
Especificación de Casos de Usos del Sistema.	G	2
Diagrama de Casos de Uso.	H	2
Descripción Textual de los CUS.	I	2
Especificación de Requerimientos.	J	2
Prototipo de Interfaz de Usuario.	K	3
Descripción de la Arquitectura (Vista de CU).	L	3
Modelo de Análisis.	M	3
Diagrama de Cases del Análisis.	N	3
Diagramas de Colaboración.	O	2
Modelo de Diseño.	P	3
Diagrama de Clases del Diseño.	Q	3
Diagrama de Secuencia.	R	2
Documento de Arquitectura (Vista de diseño).	S	3
Modelo de Despliegue.	T	2
Documento de Arquitectura (Vista de despliegue).	U	3
Modelo de Datos.	V	3
Modelo de Implementación.	W	3
Diagramas de Componentes.	X	2
Código Fuente.	Y	3
Base de Datos.	Z	3

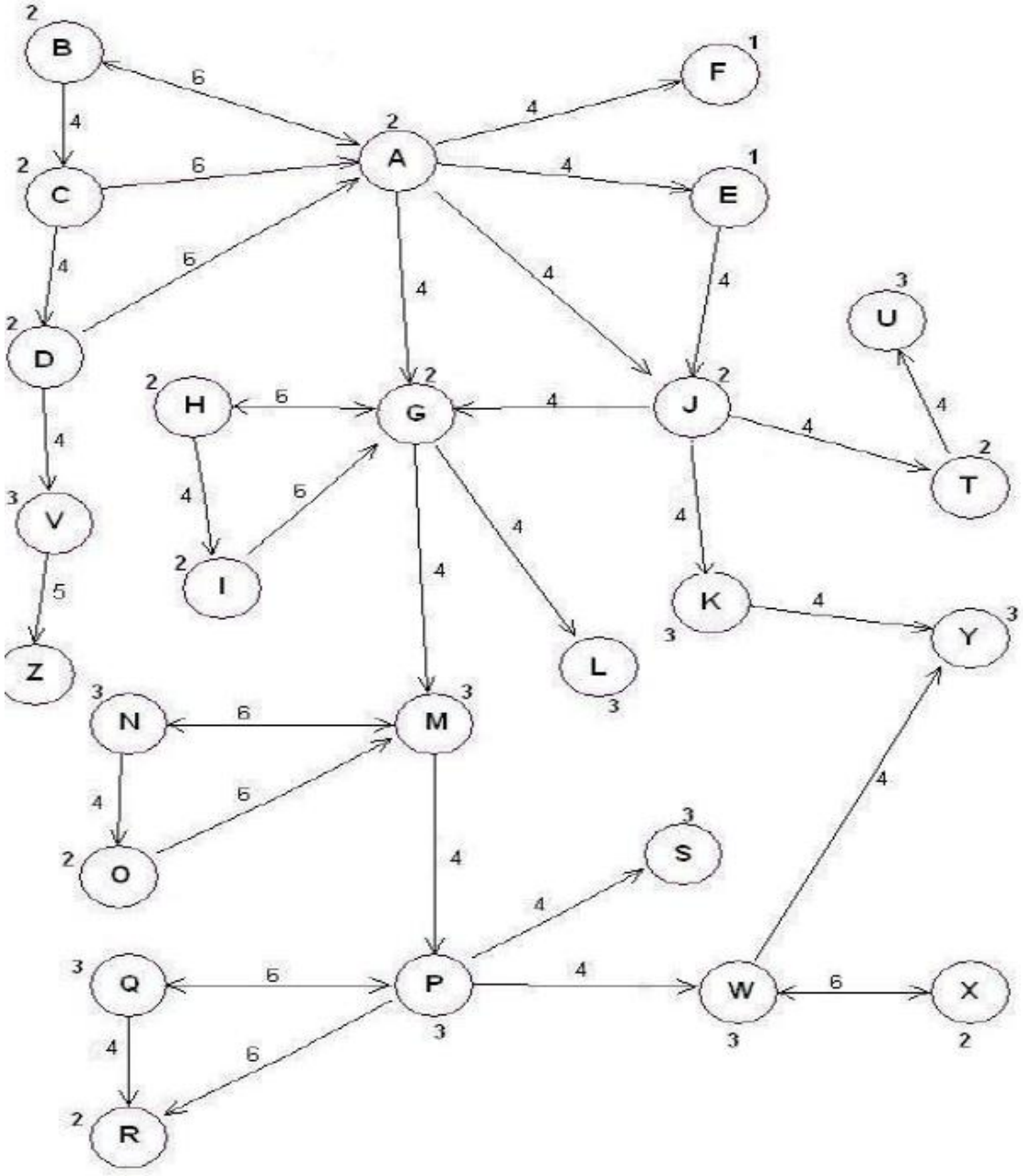
Después de haber asignado a cada ECS un nodo, es necesario expresar las relaciones que existen entre estos ECS mediante aristas ponderadas que unan a los nodos que representen a dichos ECS, para ponderar la arista debe tenerse en cuenta la *Tabla 29* donde se especifica el peso para cada tipo de relación.

Tabla 29: Asignación de Pesos a las relaciones existentes en esta Línea Base

Relación	Peso
Composición	6
Derivación	5
Dependencia	4

Realización manual Paso1

Ilustración 29: Grafo de los ECS



Luego de haber construido el grafo se procede al cálculo del alcance del cambio. Para realizar este cálculo lo primero que se hace es recorrer el grafo construido a partir del nodo que representa el ECS víctima del cambio. En este caso el Nodo que representa el Prototipo de Interfaz de Usuario es el nodo K. El recorrido quedaría de esta forma.

Tabla 30: Datos del Alcance del Nodo K

ID. Nodo	Peso Nodo(PNR)	Peso Arista(PAR)
K	3	-
Y	3	4
Σ	6	4

Para realizar los cálculos referidos al Alcance del Cambio se necesitan los siguientes datos:

(NR) = 2 (NRFR)= 1 (NTFR) = 6 (NRFI)= 1 (NTFI) = 4 (NT) = 26 (AR) = 1 (PCE) = 15 (PAG) = 153 (PNG) = 5

Complejidad de los ECS involucrados en el cambio (CC)

CC = Σ PN / NR

= 3 + 3 / 2

= 6 / 2 = 3.

Magnitud del Cambio en el proyecto (MC)

MC = (Σ PNR + Σ PAR) / (PNG + PAG)

= ((3 + 3 + 4) / (50+153))

= 10 / 203 = 0.049

Porciento del cambio (PC)

PC = PCE* (NR / NT)

= 15 * (2 / 26)

= 15 * 0.077

= 1.15

A continuación se calcula el porciento de cambio en cada una de las fases implicadas.

PCF (Requerimientos) = (NRF*100/NTF) * PCE

= (1 * 100 / 6)* 0.15 = 2.5 %

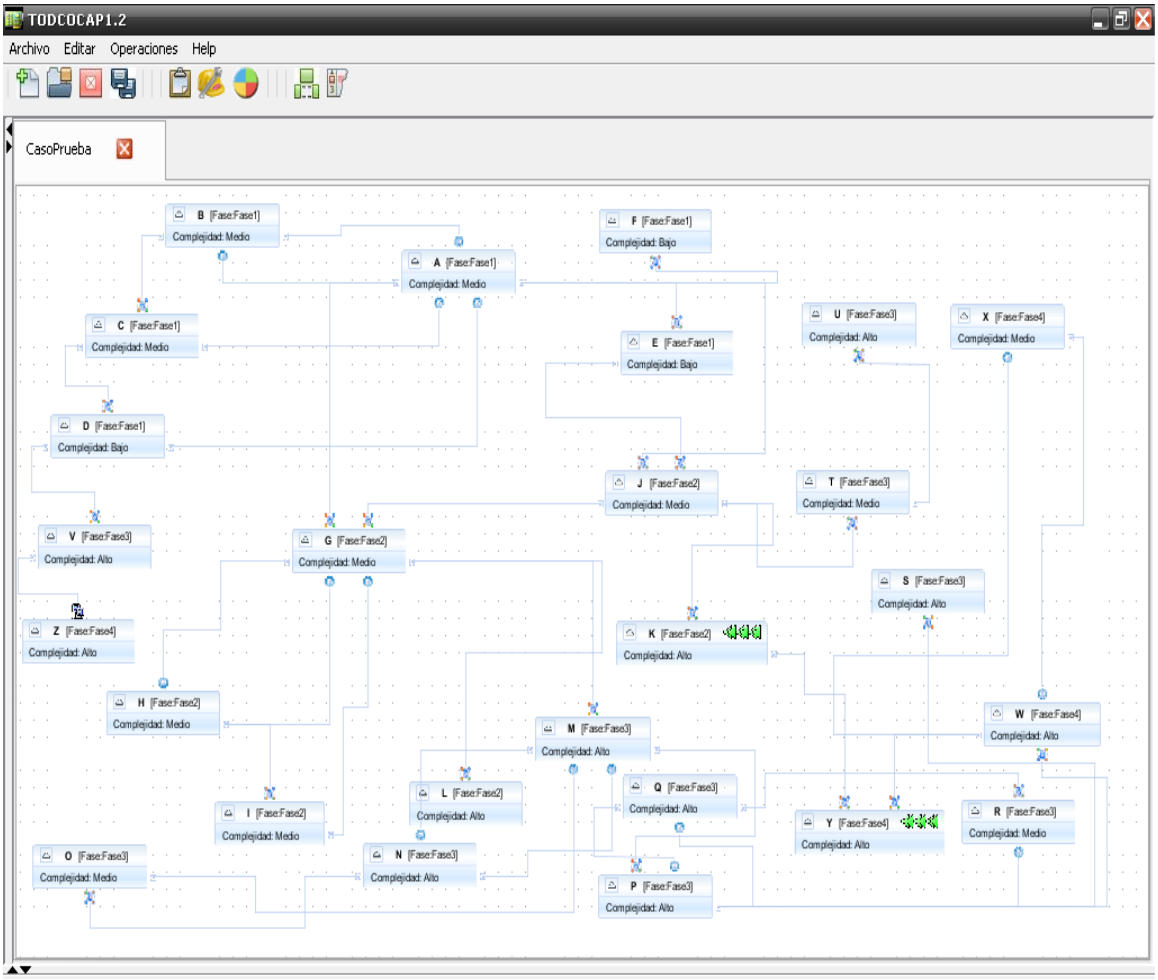
PCF (Implementación) = NRF*100/NTF * PCE

= (1 * 100 / 4) * 0.15 = 3.75%

Paso1 Con el sistema.

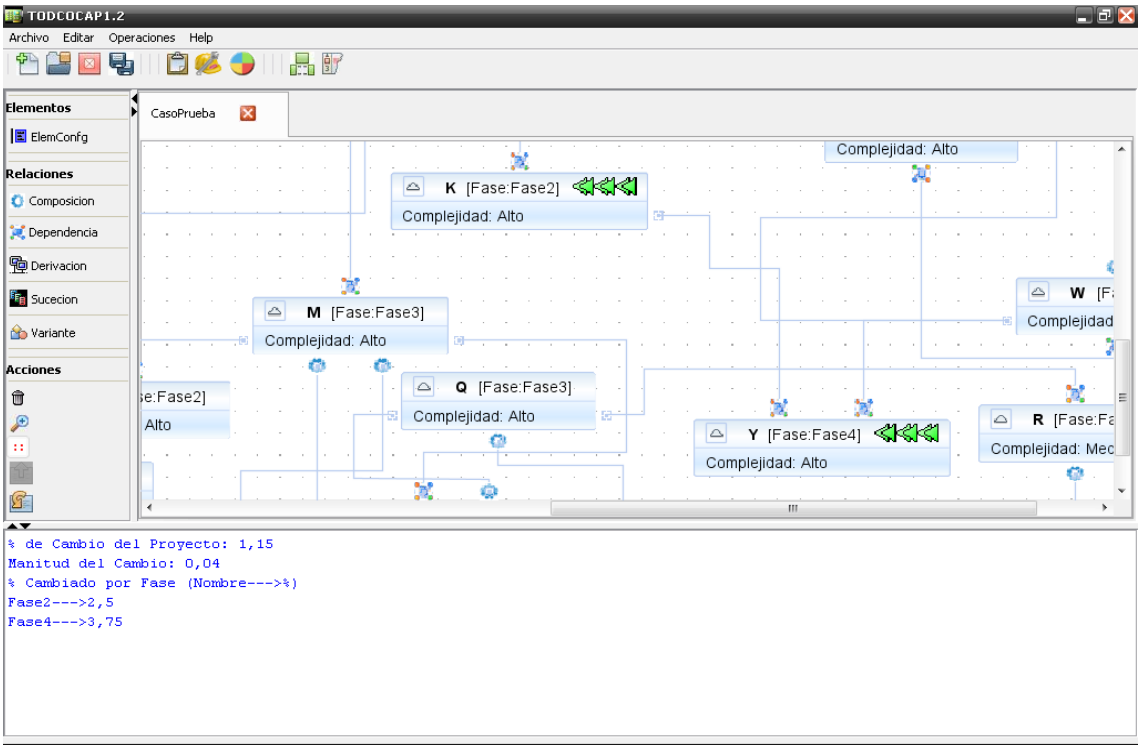
Una vez insertado el grafo quedaría de esta forma dándole el zoom adecuado para que pueda mostrarse todo el grafo en una sola pantalla.

Ilustración 30: grafo insertado



Después de hacer la entrada solo se le pide a la aplicación que muestre el reporte general y en la consola aparece lo siguiente.

Ilustración 31: Reporte general



La próxima actividad a realizar es el análisis de costo y esfuerzo que implicaría el cambio. Para realizar este análisis de debe calcular primeramente el esfuerzo que implicaría el cambio.

Realización manual Paso2.

Esfuerzo necesario para realizar los cambios en los ECS de la fase de Requerimientos (EF):

EF (Requerimientos)= PCF * ERF / 100

= (2.5* 112) / 100

= 2.8 Horas – Hombres

Esfuerzo necesario para realizar los cambios en los ECS de la fase de Implementación (EF):

EF (Implementación)= PCF * ERF / 100

= (3.75 * 192) / 100

= 7.2 Horas – Hombres.

Esfuerzo total requerido para realizar el cambio:

ET= Σ EF (Horas-Hombre)

= 2.8 + 7.2

=10

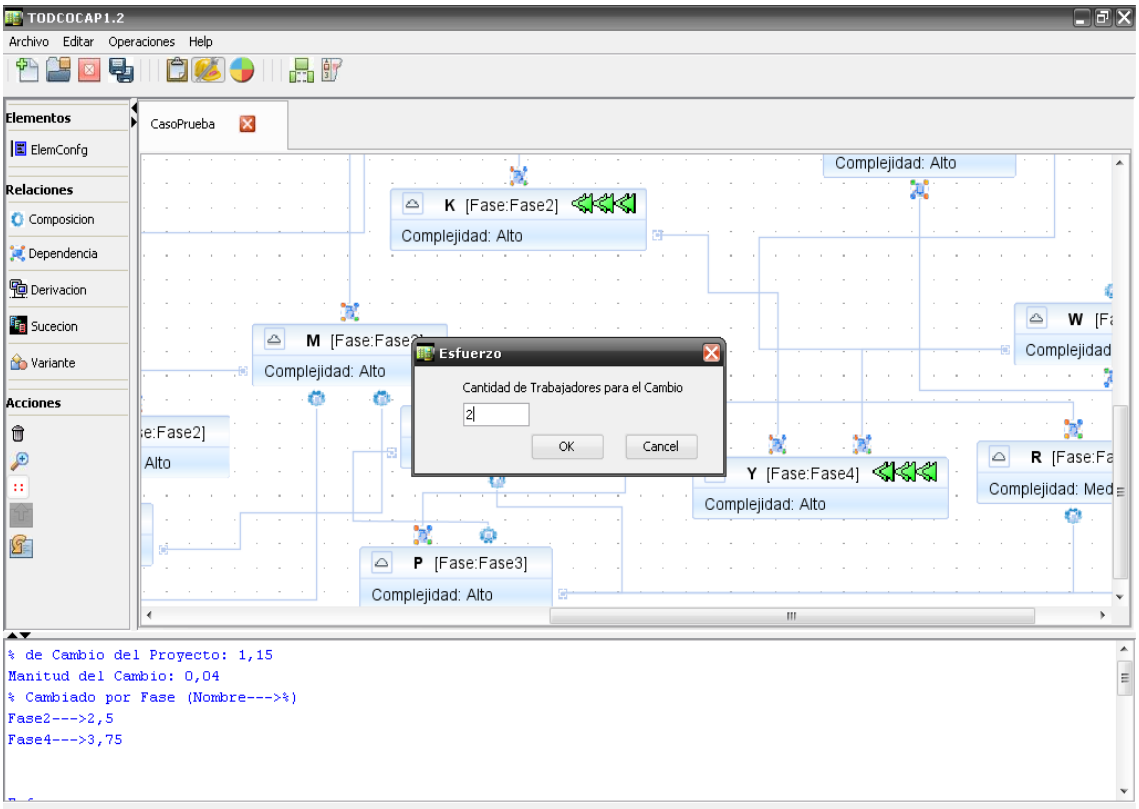
En este caso de estudio se asignarán 2 trabajadores para ejecutar el cambio.

El tiempo que se empleará en la realización del cambio en la fase de Requerimientos es de 1.4 horas y en la fase de Implementación 3.6 horas lo que representa un aproximado de 5 horas destinados para la realización del cambio.

Paso2 con el Sistema.

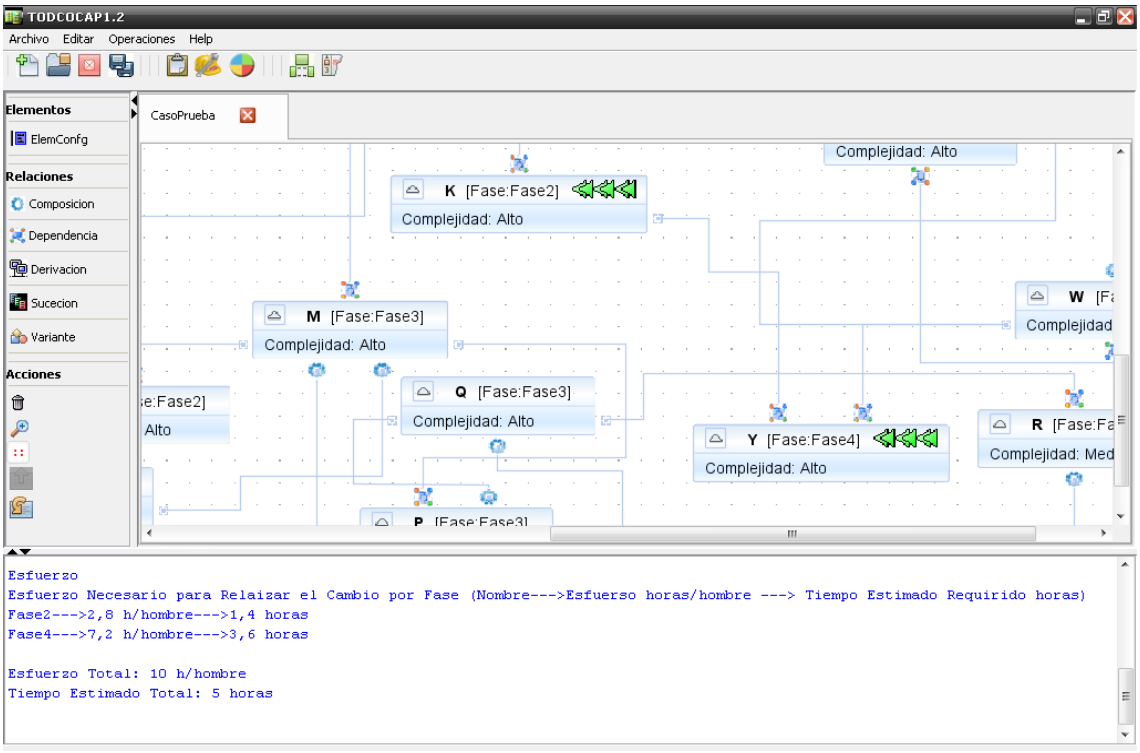
Simplemente se le pide al sistema un análisis de esfuerzo y se le introduce la cantidad de trabajadores involucrados en el cambio. Como se muestra a continuación.

Ilustración 32: análisis de esfuerzo



Una vez hecho el sistema muestra el siguiente reporte:

Ilustración 33: reporte análisis de esfuerzo



Teniendo en cuenta los trabajadores implicados en el cambio y el tiempo que estos deberán destinar a la realización del mismo, será posible calcular los gastos.

Realización Manual del Paso3.

Costo Salario: 40$.

Costo por depreciación: 2 PC * $0.50=1.00$.

Costo Corriente Eléctrica = 8 horas * $0.20 = $ 1.60

Costo Total del Cambio= 40 + 1.0 + 1.6 = $ 42.60

Presupuesto para Cambios = $ 200.

Comienzo del Cambio: 19/5/2009

Entrega el proyecto: 27/6/2009.

27.40$ < 200$

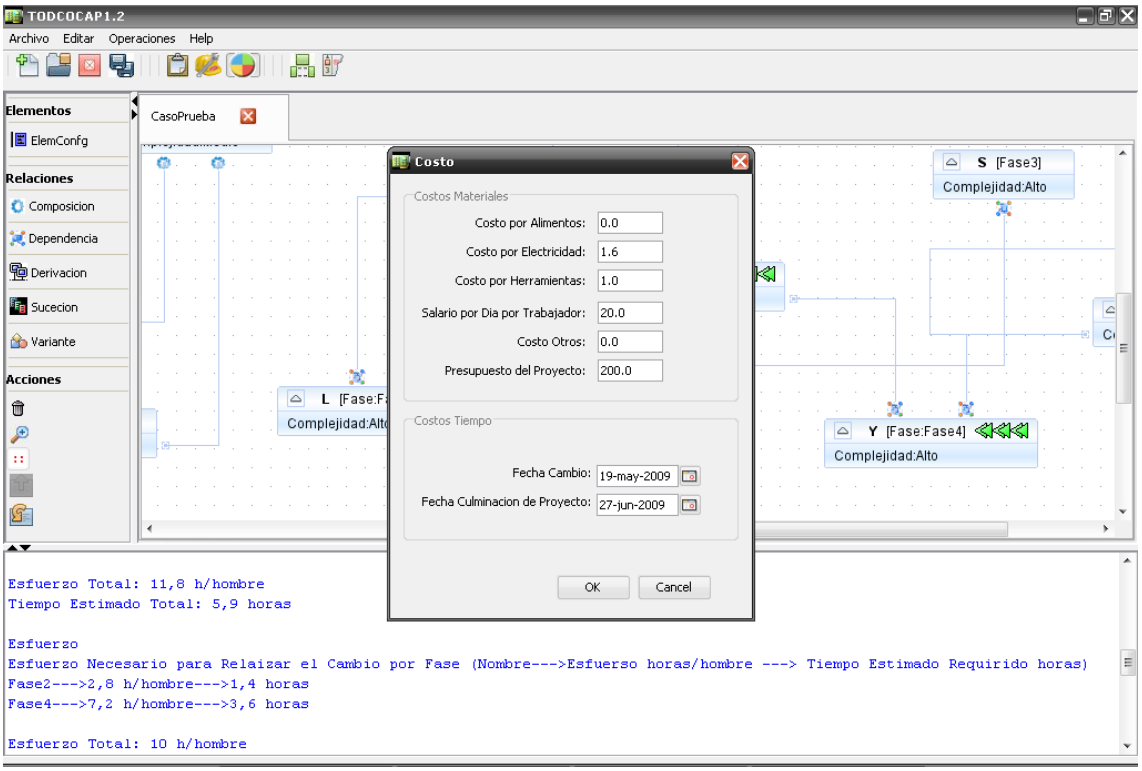
19/5/2009 + 1dia < 27/62009

Se puede realizar el cambio.

Paso 3 con el Sistema.

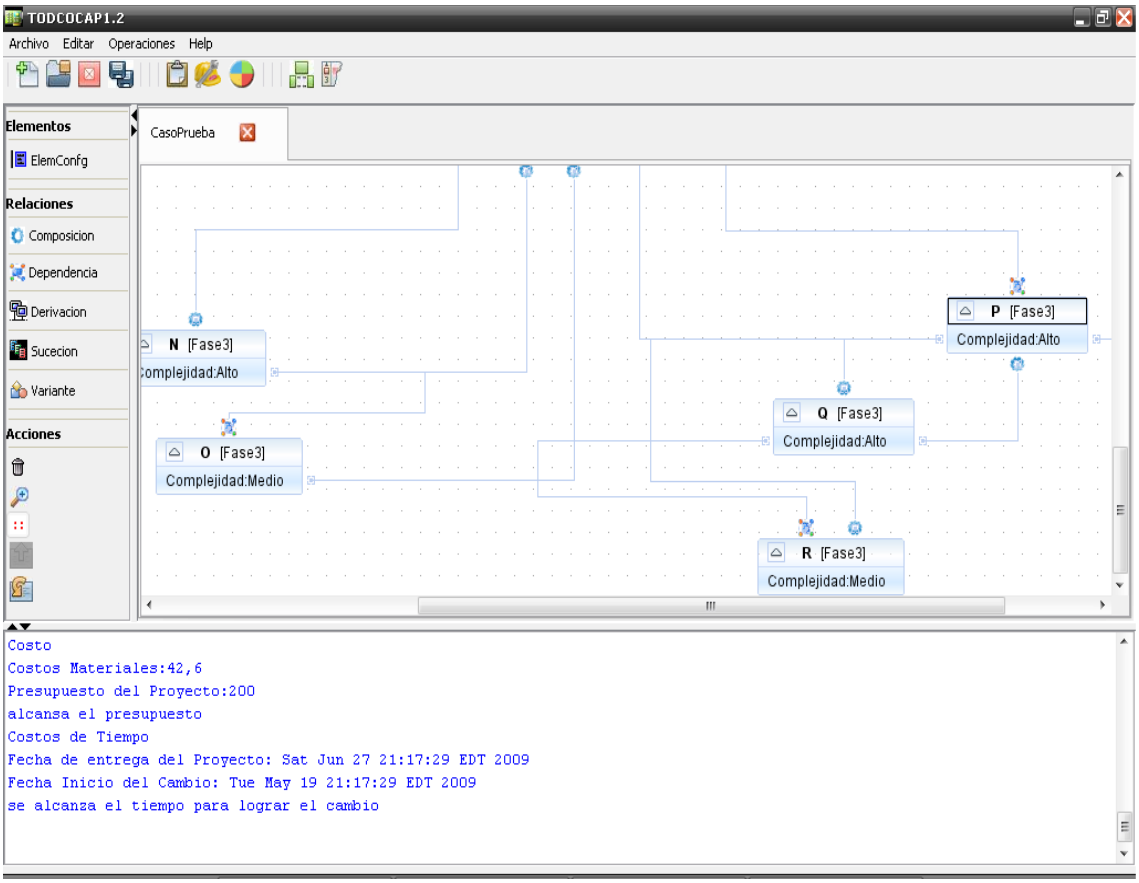
Solo seleccionando la acción costo reporte e introduciendo los datos de costo tanto materiales como de tiempo como se muestra en la figura a continuación.

Ilustración 34: análisis costo



La aplicación devuelve los resultados en la consola como se observa a continuación

Ilustración 35: reporte análisis costo



Como se puede observar el sistema abstrae completamente a los usuarios de la complejidad del procedimiento ahorrándoles en tiempo y costo. Haciendo más agradable y óptimo el uso del procedimiento en cuestión evitando errores posiblemente cometidos a la hora de realizar cálculos y eliminando lo tedioso y complejo de este.

Conclusión del Capítulo.

En el presentado capítulo se mostró y comprobó el funcionamiento específico del sistema desarrollado, mediante las pruebas de unidad, la aplicación de métricas y pruebas de interfaz. Se mostró la estructura de alto nivel de implementación. Además se presentaron las clases que se implementaron rigiéndose por las orientaciones del patrón de diseño y estándares de codificación.

Conclusiones Generales.

Este trabajo realiza una contribución al objetivo de poder estimar y controlar la productividad del trabajo en el proceso de gestión de Configuración en especifico la Toma de Decisiones en la Gestión de Cambios lo que se requiere con mucha fuerza, al estar llamada la Universidad a producir software con calidad en el menor tiempo posible. Por lo que se llevó a cabo el desarrollo de los siguientes puntos.

- Se realizó el estudio estado del arte de los procedimientos para llevar a cabo la Toma Decisiones durante el proceso de Gestión de Cambios y se decidió automatizar el que se utiliza en la Facultad 3.
- Se realizó el estudio estado del arte herramientas para llevar a cabo toma de decisiones en la Gestión de Cambios y no se encontró ninguna herramienta que satisficiera con las necesidades de la Facultad 3.
- Se realizó el estudio estado del arte de las metodologías, lenguajes y herramientas a utilizar para el desarrollo del sistema en el cual se determinó usar Java, NetBeans y como metodología de Desarrollo XP.
- Se realizó la Planificación y el Diseño del Sistema donde se generaron todos los artefactos necesarios para esas etapas en la metodología XP.
- Se realizó la Implementación y Pruebas del Sistema donde se codificó con efectividad dicha herramienta y se le realizaron diversas pruebas de calidad.
- Se realizó la validación del sistema para demostrar la efectividad de usar dicha herramienta.

Recomendaciones.

Que se utilice la herramienta propuesta en los proyectos productivos de la Facultad 3.

Que se impartan Cursos Optativos en la facultad para calificar a un grupo significativo de estudiantes que puedan en un futuro desempeñar el rol de Gestor de Cambio y llevar a cabo satisfactoriamente el proceso de toma de decisiones en la gestión de cambios utilizando dicha herramienta.

Perfeccionar el sistema de manera tal que se adecúe más a las necesidades de cualquier institución.

Extender el desarrollo de esta aplicación, adicionándole nuevas funcionalidades, extendiendo la informatización de otras tareas referentes a la Toma de Decisiones en la Gestión de Cambio.

Bibliografía

3.0, ConfigCASE. 2006. *Herramienta de apoyo a la Gestión de Configuración.* 2006.

alianze, agile. 2007. [En línea] 22 de 12 de 2007. http://www.programacionextrema.org/.

Antonio, Angélica de. 2001. *Gestion de Configuracion.* Chile : SPIN, 2001.

Appleton, Brad. 2000. SCM definitions. [En línea] 2000. http://www.enteract.com/~bradapp/scm-defs.html.

Augenstein, M, Langsam, Y. y Tenenbaum, A. 1993. *Estructuras de Datos en C.* s.l. : Brooklyn College, 1993.

Babich, W. 1986. *Software Configuration Management.* s.l. : Addison – Wesley, 1986.

Bamford, R. and Deibler, W. J. 1995. *Configuration Management and ISO 9001.* s.l. : SSQC, 1995.

Bermejo Sanz, Laura y Gómez Monreal, Enrique. 2008. Eclipse como IDE. [En línea] 2008. http://kybele.escet.urjc.es/documentos/HC/Exposiciones/EclipseIDE.pdf..

Bersoff, E. H., Henderson, V. D. y Siegel, S. G. 1980. *Software Configuration Management.* s.l. : Prentice-Hall, 1980.

Bitkeeper. 2008. [En línea] 2008. http://www.bitkeeper.com/ .

Borland. 2008. [En línea] 2008. http://www.borland.com/us/products/starteam/index.html.

Codeville. 2008. [En línea] 2008. http://codeville.org/.

Codice, Software. 2007. [En línea] 2007. http://www.codicesoftware.com/whitepapers/plastic-features-benefitsES.pdf.

Collins-Sussman, Ben, Fitzpatrick, Brian W. y Pilato, C. Michael. 2002-2008. 2002-2008.

CVS. 2008. [En línea] 2008. http://www.nongnu.org/cvs/.

CVSNT. 2008. [En línea] 2008. http://www.march-hare.com/cvsnt/es.asp.

excelsoftware. 2009. [En línea] 2009. http://www.excelsoftware.com.

Fossil. 2008. [En línea] 2008. http://www.fossil-scm.org/index.html/doc/tip.

GIT. 2008. [En línea] 2008. http://git-scm.com/.

GNU. 2008. [En línea] 21 de 05 de 2008. http://www.gnu.org/software/gnu-arch/.

González. 2007. *Estrategia de transferencia del Módulo de Factoría de Software aplicando Inteligencia Artificial.* Habana, Cuba : s.n., 2007.

Help, RUP. 2005. 2005.

IBM. 2008. [En línea] 2008. http://www-01.ibm.com/software/awdtools/clearcase/.

IEEE. 1987. *IEEE Guide to Software Configuration Management.* s.l. : American National Standards Institute, 1987. 1042-1987.

—. 1990. *IEEE Standard for Software Configuration Management Plans.* s.l. : American National Standards Institute, 1990. Std. 828-1990.

Jacobson, Ivar, Martin Griss and Patrick Johnson. 1997. *Software Reuse: Architecture, Process, and Organizartions for Business Success.* s.l. : Addison- Wesley, 1997.

Javahispano. 2009. [En línea] 2009. http://www.javahispano.org/news.item.action?id=148428245.

Joskowicz, José. 2008. *Reglas y Prácticas en.* 2008.

Larman, C. TOMO I. *UML y patrones.* TOMO I.

Marblestation. 2006. [En línea] 09 de 12 de 2006. http://www.marblestation.com/blog/?p=605.

Microsoft, Corporation. 2009. [En línea] 2009. http://msdn.microsoft.com/es-es/library/3h0544kx(VS.80).aspx.

Monotone. 2008. [En línea] 2008. http://monotone.ca/.

NetBeans. 2008. [En línea] 2008. http://netbeans.org.

Norm, Brown, Michael, W. Evans y Jeannine, Gathmann-Hobbs. 1998. *Litle Book of Configuration Manager; Computer & Concepts Associates.* 1998.

Perforce. 2008. [En línea] 2008. http://www.perforce.com/perforce/products.html.

Pressman, R. 1998. *Ingeniería de software. Un enfoque práctico.* Madrid : Mc Graw-Hill Interamericana de España S.A., , 1998.

Pressman, R.,. 2002. *Ingeniería de Software Un Enfoque Práctico. Vol. 1.* 2002.

Pressman, Roger S. 2000. *Ingeniería de Software, un enfoque práctico.* s.l. : ed. 4, Mc GrawHill Iberoamericana 2000, 2000.

Quintana Torres, Dayanis Amparo y Cid González, Yusely. 2007. *Propuesta de un Procedimiento para la Toma de Decisiones en la Gestión de Cambios de los Proyectos Productivos de la Facultad 3.* Habana : s.n., 2007.

Rational. 2003. Ofertas de precio. [En línea] 2003. http://www.indudata.com/lprecios_productos.htm#1.

Reliable, software. 2008. [En línea] 2008. http://www.relisoft.com/co_op/.

Rodríguez Corbea, Maite. 2007. *LA METODOLOGÍA XP APLICABLE AL DESARROLLO DEL SOFTWARE EDUCATIVO EN CUBA.* 2007.

Selenic. 2008. [En línea] 2008. http://www.selenic.com/mercurial.

SyBase. 2008. [En línea] 2008. http://www.sybase.com/products/modelingdevelopment/powerdesigner.

Team-Coherence. 2008. [En línea] 2008. http://www.teamcoherence.com/.

Glosario de Términos.

Artefactos: Productos tangibles del proyecto que son producidos, modificados y usados por las actividades. Pueden ser documentos, diagramas, código fuente y ejecutables.

Ciclo de vida: Se refiere al proceso que se sigue para construir, entregar y hacer evolucionar el software, desde la concepción de una idea hasta la entrega y el retiro del sistema.

ECS: Elementos de Configuración de Software

GCS: Gestión de Configuración de Software.

GC: Gestión de Cambios

Iteración: Es una secuencia de actividades con un plan establecido y criterios de evaluación, cuyo resultado es una versión del software.

Procedimiento: Sucesión cronológica de operaciones concatenadas entre sí, que se constituyen en una unidad de función para la realización de una actividad o tarea específica dentro de un ámbito predeterminado de aplicación. Todo procedimiento involucra actividades y tareas del personal, determinación de tiempos de métodos de trabajo y de control para lograr el cabal, oportuno y eficiente desarrollo de las operaciones.

RUP: Rational Unified Process.

Toma de Decisiones en la Gestión de Cambios: acto de decidir si un cambio debe ser o no realizado.

UCI: Universidad de Ciencias Informáticas.

Librería (Software): es un conjunto de subprogramas utilizados para desarrollar software.

Framework: en el desarrollo de software, es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado.

Persistir: Durar o existir una cosa durante mucho tiempo. En informática que no se pierdan una vez cerrada la aplicación.

CRC: Clase, Responsabilidades, Colaboradores.

Plug-ins: Un complemento (o plug-in en inglés) es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy especifica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la API.

Widget: En informática, un widget es una pequeña aplicación o programa, usualmente presentado en archivos o ficheros pequeños que son ejecutados por un motor de widgets o Widget Engine. Entre sus objetivos están los de dar fácil acceso a funciones frecuentemente usadas y proveer de información visual. Sin embargo los widgets pueden hacer todo lo que la imaginación desee e interactuar con servicios e información distribuida en Internet; pueden ser vistosos relojes en pantalla, notas, calculadoras, calendarios, agendas, juegos, ventanas con información del tiempo en su ciudad, etcétera.

Anexos.

Anexo 1: entrevistas realizadas en los proyectos productivos de la Facultas 3

Ilustración 36: los proyectos que realizan la Gestión de Cambio.

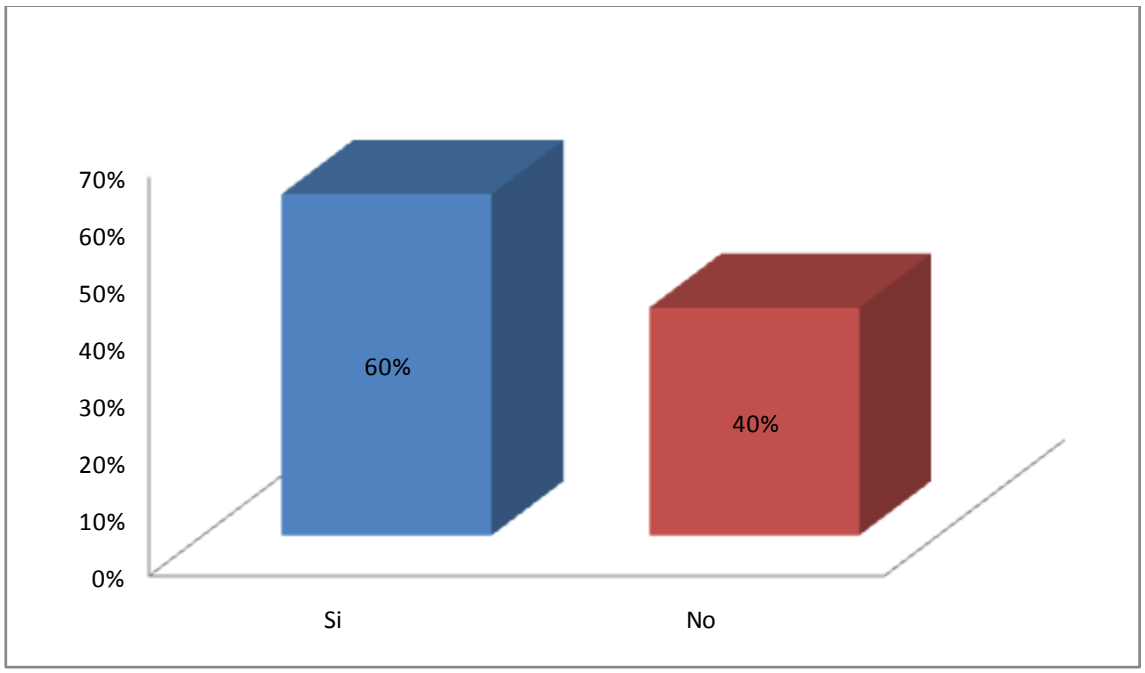


Ilustración 37: proyectos que cuentan con un procedimiento definido

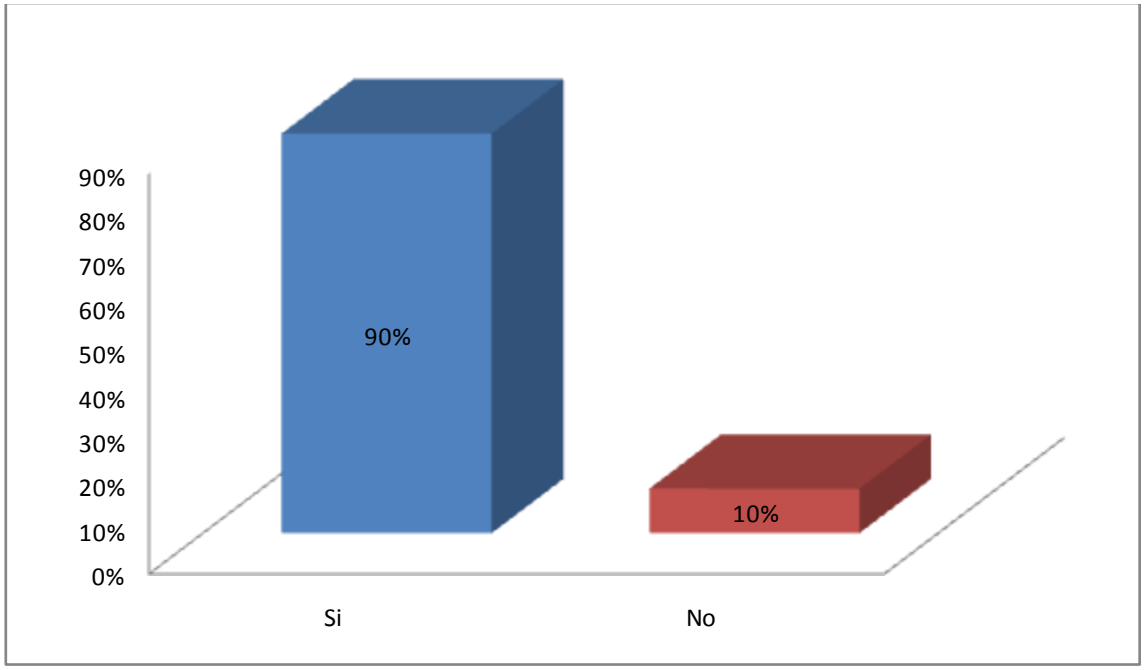
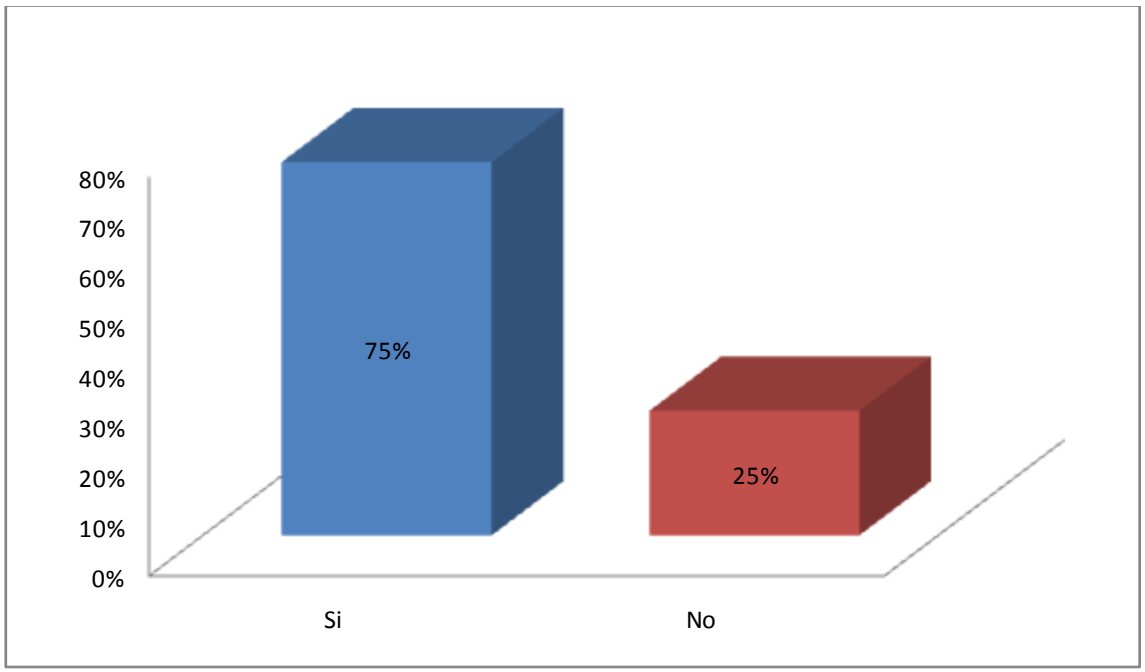


Ilustración 38: De los proyectos que cuentan con un procedimiento definido para Gestionar los Cambios realizan la actividad de Análisis del Impacto de forma empírica



Anexo 2: Tabla de precios de Plastic SCM

Tabla 31*:* Tabla de precios de Plastic SCM

Número de licencias (Edición Professional)	Precio por desarrollador (EUROS) Precio por desarrollador (US DOLLARS)
De 1 a 20	525 649
De 21 a 50	485 599
De 51 a 100	445 549
De 101 a 200	395 499
De 201 a 400	365 449
De 401 a 500	325 399
Más de 500	285 349